

“Эта книга представляет собой отличный путеводитель по бурному (и порой ошеломляющему) миру возможностей языка VBA. Простые и в то же время полезные практикумы являются чем-то вроде иллюстративных примеров, которые так важны при изучении чего-либо нового.”

— Эрика Йоксалл, Hammer Data Systems, LLC

“Гандерлой и Харкинз написали краткое и ясное руководство по освоению концепций языка VBA и его практическому применению.”

— Тим Миллз-Гронинджер, консультант
по базам данных, IT Resource Center

Бизнес-решения

Создайте свои решения

в Microsoft Access 2003,

воспользовавшись

проверенными

методиками

Автоматизация Microsoft® Access с помощью VBA

Обучение
программированию
на VBA с нуля

Использование
объектов для
сложных операций

Автоматизация
форм и элементов
управления Access

Программная
настройка отчетов

Извлечение
и обработка данных

Создание новых
объектов в программах

Выполнение
расширенных
операций с данными

Автоматизация других
приложений из Access

Импорт и экспорт
файлов XML

Майк Гандерлой
Сьюзан Сейлз Харкинз



que®

Бизнес-решения



Автоматизация Microsoft® Access с помощью VBA

business solutions



Automating Microsoft® Access with VBA

*Mike Gunderloy
Susan Sales Harkins*

QUE®

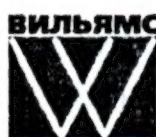
800 East 96th Street
Indianapolis, Indiana 46240

Бизнес-решения



Автоматизация Microsoft® Access с помощью VBA

*Майк Гандерлой
Сьюзан Сейлз Харкинз*



Москва • Санкт-Петербург • Киев
2006

ББК 32.973.26-018.2.75

Г19

УДК 681.3.07

Издательский дом “Вильямс”

Главный редактор *С.Н. Тригуб*

Зав. редакцией *В.Р. Гинзбург*

Перевод с английского и редакция *С.А. Храмова*

По общим вопросам обращайтесь в Издательский дом “Вильямс” по адресу:

info@williamspublishing.com, <http://www.williamspublishing.com>

115419, Москва, а/я 783; 03150, Киев, а/я 152

Гандерлой, Майк, Харкинз, Сьюзан Сейлз.

Г19 Автоматизация Microsoft Access с помощью VBA. : Пер. с англ. — М. : Издательский дом “Вильямс”, 2006. — 416 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0959-7 (рус.)

Эта книга поможет вам приумножить опыт, приобретенный при работе с приложением Access, и поднять его на принципиально новый уровень — вы научитесь использовать язык программирования VBA для выполнения тех задач, которые до сих пор делали вручную. Авторы поставили цель как можно быстрее и эффективнее научить читателей основным навыкам автоматизации работы с базами данных. В книге рассматривается большое число тем, включая: основы работы в редакторе Visual Basic Editor; использование переменных, констант, типов данных, процедур и встроенных функций VBA; работа с массивами, объектами, формами, простыми и комбинированными списками, отчетами и коллекциями приложения; анализ модели событий Access; извлечение и обработка данных с помощью ADO; работа с файлами данных и файлами XML. В конце книги дан обзор языка Jet SQL.

Книга предназначена для профессионалов, которые используют Access в своей повседневной работе.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Que Publishing.

Authorized translation from the English language edition published by Que Publishing, Copyright © 2005.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the publisher.

Russian language edition is published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2006.

ISBN 5-8459-0959-7 (рус.)

ISBN 0-7897-3244-0 (англ.)

© Издательский дом “Вильямс”, 2006

© Que Publishing, 2005

Оглавление

Введение	17
Часть I. Основы языка VBA	23
Глава 1. Почему Access? Почему VBA?	25
Глава 2. Редактор Visual Basic	35
Глава 3. Переменные, константы и типы данных	51
Глава 4. Использование процедур	65
Глава 5. Выбор правильных функций VBA	79
Глава 6. Использование инструкций передачи управления	119
Глава 7. Массивы	135
Глава 8. Объекты	143
Глава 9. Область видимости и время жизни	163
Часть II. Работа с интерфейсом пользователя Access	175
Глава 10. Работа с формами	177
Глава 11. Анализ модели событий Access	195
Глава 12. Работа с простыми и комбинированными списками	209
Глава 13. Прочие элементы управления	231
Глава 14. Работа с отчетами	247
Глава 15. Работа с коллекциями	263
Часть III. Работа с данными в Access	275
Глава 16. Извлечение данных средствами ADO	277
Глава 17. Манипуляции с данными средствами ADO	295
Глава 18. Создание объектов средствами ADOX	315
Глава 19. Расширенные операции с данными	331
Часть IV. Использование в Access расширенных средств VBA	343
Глава 20. Работа с файлами данных	345
Глава 21. Автоматизация других приложений	359

Глава 22. Работа с файлами XML	373
Глава 23. Использование Windows API	387
Приложение. Обзор языка SQL	395
Предметный указатель	408

Содержание

Об авторах	15
Посвящения	15
Благодарности	15
Ждем ваших отзывов!	16
Введение	17
Для кого эта книга	17
Как организована книга	17
Используемые соглашения	20
База данных примеров	21
Связь с авторами	22
Часть I. Основы языка VBA	23
Глава 1. Почему Access? Почему VBA?	25
Место Access в пакете Microsoft Office	25
Access или Excel?	26
Access или OneNote?	27
Понятие программирования в Access	28
Использование макросов	28
Использование языка SQL	29
Использование языка VBA	30
Глава 2. Редактор Visual Basic	35
Первый взгляд на редактор Visual Basic	35
Введение в модули VBA	38
Ввод и запуск программ на языке VBA	40
Сохранение программ	41
Получение справки по программированию	43
Полезные горячие клавиши при вводе текста программы	43
Выработка навыков по вводу кода	45
Использование соглашений об именах	46
Применение отступов в тексте программ	48
Комментарии	48
Глава 3. Переменные, константы и типы данных	51
Объявление переменных и констант	51
Объявление переменных	52
Использование оператора Option Explicit	52
Имена переменных	55
Объявление констант	55

Встроенные константы	56
Типы данных в VBA	57
Булев тип данных Boolean	59
Тип данных Byte	60
Валютный тип данных Currency	60
Тип данных Date	60
Десятичный тип данных Decimal	60
Числа с плавающей точкой двойной точности (Double)	60
Целочисленный тип данных Integer	60
Длинный целочисленный тип данных Long	60
Объектный тип данных Object	61
Тип данных Single	61
Строковый тип данных String	61
Тип данных Variant	61
Синтаксис ссылок	62
Глава 4. Использование процедур	65
Типы процедур	65
Создание и использование подпрограмм	66
Создание и использование функций	67
Объявление общих и частных процедур	69
Передача аргументов	69
Использование необязательных аргументов и значений по умолчанию	70
Передача аргументов по ссылке	71
Передача аргументов по значению	72
Присвоение функции типа данных	72
Обработка ошибок	73
Использование инструкции On Error Resume Next	73
Использование инструкции On Error Goto	74
Отладка программ	75
Использование режимов Run и Break	75
Пошаговое выполнение	76
Установка контрольных точек	77
Глава 5. Выбор правильных функций VBA	79
Встроенные функции VBA	79
Функции преобразования типов данных	80
Преобразование в булев тип данных	81
Преобразование в тип данных Byte	82
Преобразование в тип даты	83
Преобразование в целочисленный тип данных	84
Преобразование в строковый тип данных	85
Преобразование в тип данных Variant	85
Преобразование нулевых значений	85
Функции работы с датами	88
Возвращение даты	88

Операции сложения и вычитания дат	89
Вычисление разницы между двумя датами	89
Извлечение компонентов даты	91
Создание даты из отдельных компонентов	91
Создание даты из строкового выражения	92
Извлечение заданного компонента даты и времени	93
Математические и финансовые функции	94
Функция Abs	94
Функция Int	94
Функция Rnd	95
Функция Ddb	96
Функция Fv	96
Функция IPmt	97
Функция NPer	97
Функция Pmt	97
Функция PPmt	98
Функция Rate	98
Функция Syd	98
Функции работы со строками	100
Функция Asc	100
Функция Chr	101
Функции изменения регистра	102
Функция Len	102
Функции Left, Right и Mid	102
Функция Replace	103
Функция Space	103
Функция Split	103
Функция StrComp	104
Функции удаления пробелов	104
Использование функции Format	105
Применение форматов, определенных пользователем	107
Использование функций семейства Is для беспроблемного выполнения программ	110
Функции взаимодействия с пользователем	111
Функция InputBox	112
Функция MsgBox	112
Глава 6. Использование инструкций передачи управления	119
Циклы и разветвления	119
Конструкция If...Then...Else	120
Простая инструкция If	120
Создание более сложных условий	120
Добавление инструкции Else	121
Использование инструкции ElseIf	122
Конструкция Select Case	122
Конструкция цикла For...Next	123

Циклы с обратным отсчетом	125
Использование для счетчика цикла переменных границ	126
Вложенные циклы For...Next	126
Выход из цикла For...Next	127
Конструкция цикла Do	128
Простой цикл Do	128
Разновидности циклов Do	129
Выход из цикла Do	130
Безусловный переход GoTo	130
Глава 7. Массивы	135
Введение в переменные массивов	135
Объявление переменных массивов	136
Понятие индекса массива	136
Инструкция Option Base	137
Работа с элементами массивов	138
Определение элементов массива	138
Ссылка на элементы массива	139
Многомерные массивы	140
Динамические массивы	141
Инструкция ReDim	141
Глава 8. Объекты	143
Понятие объекта	143
Ссылка на реальный мир	143
Пример объекта Access	144
Создание объектов в тексте программы	144
Чтение и установка свойств	146
Вызов методов	148
Работа с коллекциями	150
Объектная модель	152
Использование объектной модели	152
Использование ссылок	153
Обозреватель объектов	154
Создание собственных объектов	155
Работа с событиями	158
Глава 9. Область видимости и время жизни	163
Что такое область видимости	163
Переменные уровня процедуры	164
Переменные и константы уровня модуля	165
Глобальные переменные и константы	166
Время жизни переменных и констант	168
Время жизни переменных уровня процедуры	169
Время жизни переменных уровня модуля	170
Время жизни общих переменных	171
Статические переменные	172

Часть II. Работа с интерфейсом пользователя Access	175
Глава 10. Работа с формами	177
Открытие и закрытие форм	177
Открытие форм	177
Закрытие формы	178
Модуль формы и обработка событий	179
Выполнение часто встречающихся задач	180
Проверка существования формы	180
Проверка факта загрузки формы	182
Перемещение формы и изменение ее размеров	182
Передача данных в форму через аргумент OpenArgs	184
Наполнение форм содержанием	185
Обработка ошибок на уровне формы	188
Работа с несколькими экземплярами формы	190
Глава 11. Анализ модели событий Access	195
Отклик на события	195
Порядок наступления событий в элементах управления	197
События переключения фокуса	197
События, связанные с данными	200
События, специфичные для элементов управления	202
Последовательность событий в формах	202
События навигации	203
События работы с данными в форме	203
За кулисами: буферы данных	204
Последовательность событий в отчетах	205
Отмена событий	206
Глава 12. Работа с простыми и комбинированными списками	209
Заполнение элементов списка	209
Простой элемент с фильтрованным списком	211
Добавление и не добавление в список	215
Обновление списка значений	216
Обновление списка при связи с таблицей или запросом	218
Множественный выбор	222
Как определить, что выбрано, а что нет	223
Функции обратного вызова	225
Глава 13. Прочие элементы управления	231
Работа с текстовыми полями	231
Ключевые свойства текстовых полей	231
Отслеживание фокуса	233
Работа со свободными текстовыми полями	235
Использование элементов в группах переключателей	236
Работа с подформами	238

Работа со свойством Tag (Дополнительные сведения)	239
Глава 14. Работа с отчетами	247
Модули и события отчетов	247
Открытие и закрытие отчетов	248
Открытие отчетов	248
Закрытие отчета	250
Передача данных в отчет через аргумент OpenArgs	250
Наполнение отчета данными	252
Применение фильтра и сортировки	253
Обработка ошибок уровня отчета	254
Что делать при отсутствии данных?	256
Использование VBA для определения свойств группировки в отчете	257
Глава 15. Работа с коллекциями	263
Коллекции Access	263
Получение списка объектов	265
Работа со свойствами объектов	266
Программно определяемые зависимости	269
Часть III. Работа с данными в Access	275
Глава 16. Извлечение данных средствами ADO	277
Что такое библиотека ADO и зачем она нужна	277
Объектная модель ADO	278
Использование объекта ADO Connection	278
Открытие подключения	279
Строка подключения	281
Закрытие подключения	282
Работа с объектами Command	283
Создание объекта Command	283
Выполнение команд	284
Различные типы наборов записей	285
Создание и открытие набора записей	286
Фильтрация набора записей	287
Использование свойства Recordset	289
Глава 17. Манипуляции с данными средствами ADO	295
Перемещение по набору данных	295
Ссылка на поля набора данных	298
Поиск данных в объекте Recordset	299
Альтернатива в поиске — метод Seek	301
Добавление данных в объекте Recordset	303
Удаление данных в объекте Recordset	305
Обновление данных в объекте Recordset	306
Использование транзакций для изменения группы записей	309

Глава 18. Создание объектов средствами ADOX	315
Создание таблиц	316
Создание таблицы и ее столбцов	317
Создание индексов	319
Создание отношений	320
Защита объектов	321
Создание новой группы	322
Создание новых пользователей	323
Изменение владельца объекта	323
Установка прав доступа к объектам	325
Глава 19. Расширенные операции с данными	331
Программирование конкуренции	331
Понятие конкуренции	331
Оптимистическая блокировка в ADO	333
Пессимистическая блокировка в ADO	336
Извлечение пользовательского набора данных	337
Использование других наборов данных схем	339
Часть IV. Использование в Access расширенных средств VBA	343
Глава 20. Работа с файлами данных	345
Понятие операций файлового ввода-вывода	345
Открытие файлов	346
Аргумент <i>режим</i>	346
Аргумент <i>доступ</i>	347
Аргумент <i>блокировка</i>	347
Простой пример открытия файла	348
Чтение из файлов	349
Функция Input	349
Инструкция Line Input #	351
Инструкция Input #	351
Запись в файл	353
Печать в файл	354
Глава 21. Автоматизация других приложений	359
Понятие автоматизации	359
Создание ссылок на объекты	360
Создание объектов на сервере автоматизации	362
Использование функции CreateObject	362
Использование функции GetObject	363
Использование раннего связывания	364
Взаимодействие с приложением Excel из Access	365
Взаимодействие с приложением Word из Access	367

Глава 22. Работа с файлами XML	373
Введение в XML	373
Использование метода ExportXML	374
Пример экспорта	376
Экспорт файлов для Web	377
Экспорт связанных данных	378
Использование метода ImportXML	379
Пример импорта	380
Глава 23. Использование Windows API	387
Объявление вызовов API	387
Использование вызовов API	389
Вызовы API, которые можно использовать в Access	390
Как определить, запущено ли приложение	390
Получение имени текущего пользователя	391
Получение имени программы обработки для файла данных	392
Когда следует использовать Windows API	393
Приложение. Обзор языка SQL	395
Введение в SQL	395
Структура и синтаксис языка SQL	396
Извлечение данных с помощью инструкции SELECT	398
Предикаты SQL	399
Предложение FROM	400
Предложение WHERE	400
Предложение ORDER BY	401
Предложение GROUP BY	402
Предложение HAVING	403
Изменение данных с помощью инструкции UPDATE	403
Удаление записей с помощью инструкции DELETE	404
Добавление записей с помощью инструкции INSERT INTO	405
Создание таблиц с помощью инструкции SELECT INTO	406
Создание перекрестного запроса с помощью инструкции TRANSFORM	406
Предметный указатель	408

Об авторах

Майк Гандерлой (Mike Gunderloy) — независимый разработчик и автор, работающий с компьютерами уже на протяжении 25 лет. Его опыт работы с пакетом Microsoft Office начался с версии 4.3, которая, несмотря на свой номер, по сути была первой версией этого интегрированного пакета. Все последующие годы Майк тесно сотрудничал с командой разработчиков пакета Microsoft Office, и даже написал ряд программных модулей, интегрированных в этот пакет. Из-под пера Майка вышло более 20 книг, посвященных вопросам программирования. В настоящее время он работает редактором еженедельника *Developer Central*. Связаться с Майком можно по электронному адресу MikeG1@larkfarm.com. При желании вы можете посетить любой из его Web-узлов: <http://www.larkware.com> или <http://www.codertodeveloper.com>.

Сьюзан Сейлз Харкинз (Susan Sales Harkins) — независимый консультант и эксперт в области программы Access. Вместе с Майком Сьюзан выпустила книгу *Upgrader's Guide to Microsoft Office System 2003*. В настоящее время она пишет статьи в различные технические журналы, газеты и электронные издания, в том числе в *Element K Journals*, *builder.com* и *devx.com*. Среди последних работ, изданных в соавторстве с Майком, книги *Exam Cram 2 ICDL*, *ICDL Practice questions Exam Cram 2*, *Absolute Beginner's Guide to Access 2002* и *Absolute Beginner's Guide to Access 2000*. Все эти книги вышли в издательстве Que.

Посвящения

Посвящается Томасу, родители которого были заняты выпуском этой книги.

Майк Гандерлой

Посвящается Лекси, который помогает мне оставаться молодой, и Биллу, помогающему мне закрепиться в жизни.

Сьюзан Харкинз

Благодарности

Во время завершения работы над рукописью автору предстоит решить самую приятную и в то же время непростую задачу — поблагодарить всех тех людей, которые помогли издать книгу. Лоретта Ейтс выдвинула идею выпуска данной книги в новой серии. Большую помощь оказали нам и другие члены редакторской команды: Сонглин Кью, Джордж Недефф и Дана Джонс. Хотелось бы также поблагодарить полиграфический коллектив издательства, который сумел воплотить наши мысли и идеи в жизнь.

Возможно, в книгу все-таки вкрались какие-то ошибки и опечатки, но это произошло вопреки желанию всех этих людей. Мы стремились сделать книгу именно такой, какой ее желаете (по нашему мнению) видеть вы, и мы выражаем вам благодарность за ее покупку.

Майк благодарен своей семье, которая во время написания им очередной книги взвалила на свои плечи все домашние заботы. Дана в дополнение к своей основной работе занималась поддержанием порядка в доме, исправлением моих опечаток и грамматических ошибок, не говоря уже о воспитании ребенка. Адам и Кайла поддерживали меня своими улыбками и крепкими объятиями, и как бы прощали меня за недостаток внимания. А Томас имел достаточную выдержку, чтобы не появиться на свет, пока последняя глава рукописи не была закончена.

Сьюзан хотела бы выразить благодарность всему коллективу издательства “Que” за поддержку инновационных книжных проектов и за постоянное внимание к процессу работы над этими проектами. Особую благодарность Сьюзан выражает своей семье, которая поддержала ее в решении работать дома.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится вам эта книга или нет, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: info@williamspublishing.com

WWW: <http://www.williamspublishing.com>

Адреса для писем:

из России: 115419, Москва, а/я 783

из Украины: 03150, Киев, а/я 152

Введение

Книга *Автоматизация Microsoft Access с помощью VBA* поможет вам усовершенствовать навыки, приобретенные при работе с Access, и применять их на принципиально новом уровне — вы научитесь использовать язык программирования для выполнения тех задач, которые до сих пор делали вручную. Приложение Access позволяет использовать язык программирования, названный *Visual Basic for Applications (VBA)*, и даже если вы никогда ранее не занимались программированием, с его помощью вы сделаете свою работу в Access более продуктивной.

Для кого эта книга

Эта книга предназначена для профессионалов, которые в своей повседневной работе используют Access. Естественно, мы не предполагаем, что вы являетесь профессиональным программистом, но в то же время допускаем, что за компьютером вы проводите большую часть рабочего времени. Естественно, вы — занятая личность и не можете посвятить дни напролет чтению компьютерных книг (это существенно ограничило наши рамки). Мы поставили себе целью как можно быстрее и эффективнее научить читателя основным навыкам автоматизации работы с базами данных, поэтому даже если вы только начинающий программист, то довольно быстро ощутите результат после прочтения нескольких глав. В настоящей книге мы постарались осветить как можно больше различных методик (безусловно, к некоторым из них вы будете обращаться чаще, чем к другим).

При написании книги была использована программа Access 2003, входящая в офисный пакет Microsoft Office 2003. Однако если у вас установлена более старая версия этого пакета, вам не следует волноваться и стараться любой ценой ее обновить, поскольку все описанные здесь функции и средства языка VBA не изменялись на протяжении нескольких последних лет. Все описанные в книге примеры одинаково работоспособны в версиях Access 2002 и Access 2000, хотя внешний вид некоторых элементов интерфейса может в ранних версиях приложения немного отличаться.

Как организована книга

Книга поделена на 23 главы, организованные в четыре части, и имеет одно приложение.

- Первая часть, “Основы языка VBA”, знакомит вас с синтаксисом и структурами программирования, которые обязательно знать для самостоятельного программирования на этом языке.

Глава 1 “Почему Access? Почему VBA?” позволит вам сориентироваться в материале книги. Access — не единственное приложение пакета Microsoft Office, в котором можно программировать на языке VBA. В то же время сам язык VBA нельзя назвать единственным инструментом программирования в офисных приложениях. Прочитав первую главу, вы поймете, почему именно этот язык считается лучшим для автоматизации работы в программе Access.

В главе 2 “Редактор Visual Basic” представлен пользовательский интерфейс редактора, используемого для написания программ на языке VBA. Здесь же будут раскрыты некоторые полезные приемы написания программного кода.

В главе 3 “Переменные, константы и типы данных” представлен первый набор концепций, которые нужно понять для написания программ на VBA.

В главе 4 “Использование процедур” описаны средства организации программного кода на языке VBA. Процедурами называют независимые фрагменты текста программы, которые можно выполнять в процессе работы как единое целое. По ходу изложения материала настоящей книги будет описано множество процедур.

В главе 5 “Выбор правильных функций VBA” описан ряд встроенных инструментов языка VBA, позволяющих выполнять операции над датами, финансами и текстом, а также реализующих множество математических функций. Использование этих функций поможет вам сэкономить время, затраченное на написание собственного кода.

В главе 6 “Использование инструкций передачи управления” продемонстрированы средства, которые в языке VBA позволяют выполнять разные действия при различных исходных условиях. (К примеру, можно написать процедуру, которая выполняет одни действия при передаче ей положительного числа, и совершенно другие — при передаче отрицательного.)

В главе 7 “Массивы” описаны способы хранения в одной переменной целых массивов информации. Массивы уместно использовать при работе с большими группами однородных объектов.

В главе 8 “Объекты” рассматриваются некоторые особо важные концепции языка VBA. Объекты позволяют создавать в программах структуры, полностью описывающие разнородные характеристики какого-либо предмета. Множество встроенных объектов языка VBA и программы Access можно использовать в тексте собственных программ и описывать такие объекты, как, например, экранные формы.

В главе 9 “Область видимости и время жизни” рассматривается ряд методов управления переменными в Access.

- Во второй части, “Работа с интерфейсом пользователя Access”, подробнее раскрываются основные понятия, описанные в первой части, и рассматриваются способы работы с интерфейсом пользователя программы Access непосредственно из текста программы. Все, что в этом приложении вы можете делать вручную, от открытия форм до запуска запросов на выполнение, можно сделать и программным путем. Именно здесь язык VBA позволяет автоматизировать работу с Access, заменяя ручные действия выполнением сценариев.

В главе 10 “Работа с формами” будет показано, как с помощью VBA автоматизировать формы в Access. Вы узнаете, как открывать и закрывать формы, как открывать множество экземпляров одной и той же формы и как передавать в них информацию.

В главе 11 “Анализ модели событий Access” вы познакомитесь с процессом обработки событий в Access. События позволяют запускать на выполнение определенные программы, когда на экране что-то происходит. К примеру, некоторый фрагмент кода можно прикрепить к форме так, чтобы он выполнялся при каждом ее открытии.

В главе 12 “Работа с простыми и комбинированными списками” показано, как на языке VBA описывать и использовать эти два важных экранных элемента.

В главе 13 “Прочие элементы управления” показано, как в программах на языке VBA описывать и использовать множество разнообразных экранных элементов управления, в том числе текстовые поля, переключатели и подформы.

Глава 14 “Работа с отчетами” продемонстрирует вам, как с помощью программ на языке VBA создавать в Access отчеты. Программы позволяют практически полностью управлять данными и их компоновкой в отчетах Access.

В главе 15 “Работа с коллекциями” вы узнаете, как использовать VBA для получения информации о разных объектах приложения Access, в том числе о формах, отчетах, таблицах и запросах.

- Третья часть, “Работа с данными в Access”, переносит вас от экранных элементов интерфейса к информации, хранящейся в базах данных. Здесь вы узнаете, как использовать функции библиотеки ADO (ActiveX Data Objects) для чтения и изменения данных.

В главе 16 “Извлечение данных средствами ADO” вы узнаете, как извлекать данные из таблиц и запросов и помещать их в объекты динамических наборов данных в памяти. Вы узнаете, как получать те данные, которыми будет оперировать программа в процессе своего выполнения.

В главе 17 “Манипуляции с данными средствами ADO” описаны процессы добавления, удаления и обновления данных. С использованием средства библиотеки ADO все эти операции выполнять совсем несложно.

В главе 18 “Создание объектов средствами ADOX” описан особый раздел библиотеки ADO, позволяющей создавать собственные объекты хранения данных. К примеру, с помощью отдельных функций можно создавать новые таблицы, даже не прикасаясь к интерфейсу пользователя приложения.

В главе 19 “Расширенные операции с данными” описываются дополнительные операции, которые позволяет выполнять библиотека ADO. В главе рассматривается вопрос одновременной работы с одной базой данных нескольких пользователей.

- Четвертая часть, “Использование в Access расширенных средств VBA”, посвящена продвинутым средствам языка VBA, которые демонстрируют истинную его мощь.

В главе 20 “Работа с файлами данных” демонстрируются методики работы с данными, хранящимися в обычных текстовых файлах. Несмотря на то, что программа Access хранит собственную информацию в базах данных, с помощью языка VBA вы получаете возможность работы и с другими типами файлов.

В главе 21 “Автоматизация других приложений” показано, как в программе Access использовать программы для манипуляции другими приложениями, такими как Microsoft Word и Microsoft Excel. Вы убедитесь, насколько легко использовать эти технологии для привлечения функций из других приложений.

В главе 22 “Работа с файлами XML” рассматривается поддержка языком VBA и программой Access файлов специальной структуры — XML (Extensible Markup Language). Файлы XML в настоящее время широко используются в компьютерной индустрии, так что рано или поздно вы все равно с ними столкнетесь.

Завершает книгу глава 23 “Использование Windows API”, которая посвящена работе с функциями, заложенными в самой операционной системе Microsoft Windows.

- В приложении “Обзор языка SQL” представлен обзор использования языка SQL для извлечения и обработки данных в программе Access. Несмотря на то, что SQL не является составной частью языка VBA, его знание необходимо для понимания некоторых вопросов, освещаемых в настоящей книге.

Используемые соглашения

В настоящей книге использован ряд средств и соглашений, которые облегчают восприятие материала и помогают эффективно и максимально быстро научиться работать с программой.

- Строки программ, команды, операторы, переменные и комментарии отображаются моноширинным текстом.
- *Курсивом* выделяются технические термины, определения которых приводятся в материале.
- При описании синтаксиса операторов и функций *курсив* также используется для аргументов. Эти аргументы при вызове функций в программе заменяются реальными данными и переменными. Заключенные в квадратные скобки аргументы являются в функциях не обязательными.

Важная информация, на которую следует особо обратить внимание, выделена в книге отдельными блоками.

→ Перекрестные ссылки показывают, где в книге содержится информация, связанная с рассматриваемым вопросом.

Предупреждение

Блоки-предупреждения сообщают о скрытых подводных камнях процесса выполнения рассматриваемых задач. При работе с компьютером всегда существует опасность потери данных. Эти блоки помогут вам избежать некоторых ловушек.

Примечание

Блоки примечаний содержат информацию, на которую следует обратить особое внимание. Они помогут вам лучше разобраться в теме.

Совет

Блоки советов подскажут вам методы работы, которые легче, быстрее и эффективнее стандартных.

Некоторые главы книги заканчиваются практикумами. В них содержатся расширенные примеры, которые демонстрируют средства и методики, описанные в главе, в применении к реальным производственным ситуациям. Эти практикумы помогут читателю усвоить материал, посвященный языку VBA, и уверенно идти по пути автоматизации собственных баз данных.

База данных примеров

Во всех практикумах, которые приведены в настоящей книге, используется база данных TimeTrack. В начале книги эта база не содержит никаких средств автоматизации. В ней отслеживается время, затраченное сотрудниками компании на обслуживание клиентов в рамках выполнения отдельных проектов. Структура этой базы данных подробно описана в главе 1.

Примеры программ каждой из глав помещены в базу данных TimeTrack в виде отдельных модулей. К примеру, модуль Chapter 5 содержит программы

из пятой главы книги. К тому времени, когда вы дочитаете книгу до конца, вы убедитесь на собственном опыте, какие неоценимые блага приносит автоматизация баз данных.

Базу данных TimeTrack можно загрузить с Web-странички книги на сайтах www.williamspublishing.com (русифицированная версия) и querpublishing.com (исходная версия).

Связь с авторами

Авторам всегда интересно узнать мнение читателей. Мы не сможем помочь вам в обновлении пакета Microsoft Office, но все же будем рады получить ваши отзывы, замечания и рекомендации. Со Сьюзан вы можете связаться по адресу ssharkins@bellsouth.net, а с Майком — по адресу MikeG1@larkfarm.com.

ОСНОВЫ ЯЗЫКА VBA

- 1. Почему Access? Почему VBA?**
- 2. Редактор Visual Basic**
- 3. Переменные, константы и типы данных**
- 4. Использование процедур**
- 5. Выбор правильных функций VBA**
- 6. Использование инструкций передачи управления**
- 7. Массивы**
- 8. Объекты**
- 9. Область видимости и время жизни**

Почему Access? Почему VBA?

1

Место Access в пакете Microsoft Office

Добро пожаловать в книгу *Автоматизация Microsoft Access с помощью VBA!* Эта книга поможет вам начать использовать базы данных Access не только в качестве хранилища информации. Язык *VBA* (Visual Basic for Applications) лежит в основе средств автоматизации, встроенных в программу Microsoft Access. Вы узнаете, как повысить скорость ввода данных, выполнять сложные вычислительные производственные задачи и даже обмениваться данными с другими приложениями. Даже если вы никогда не сталкивались с программированием, прочтя эту книгу, вы начнете составлять программы не хуже истинного профессионала.

Однако перед тем как окунуться с головой в мир автоматизации процессов в программе Access, несколько страниц мы посвятим описанию места программы Access и языка VBA в пакете Microsoft Office, в частности в структуре средств его автоматизации. Именно этому вопросу будет посвящена первая глава книги. Несмотря на то, что Access и VBA — довольно популярные и многосторонние инструменты, их нельзя назвать универсальными средствами решения абсолютно всех задач, поэтому вам следует знать о существовании альтернативных решений.

В ЭТОЙ ГЛАВЕ

Место Access в пакете Microsoft Office	25
Понятие программирования в Access	28

В книге предполагается, что вы уже используете (или, по крайней мере, собираетесь использовать) базы данных Access для хранения информации. В то же время пакет Microsoft Office 2003 для хранения информации предлагает три совершенно разных приложения:

- Microsoft Office Access 2003;
- Microsoft Office Excel 2003;
- Microsoft Office OneNote 2003.

Первой нашей задачей является выбор наилучшего приложения для хранения данных конкретного типа.

Access или Excel?

Поскольку оба этих приложения создавались для хранения и выполнения операций с деловой информацией, самым сложным для пользователя является выбор именно между этими программами.

Электронные таблицы окружали нас еще со времен появления компьютеров; они предлагают привычный и удобный интерфейс для хранения информации. С другой стороны, работа в Access может представлять несколько большую сложность, поскольку базы данных имеют более сложную организацию, чем электронные таблицы. Это разделение проявляется также и в составе пакета Microsoft Office — в недорогие версии программа Access просто не включается.

Несмотря на то, что Access и Excel хранят данные в таблицах, состоящих из строк и столбцов (рис. 1.1), между ними существуют серьезные конструктивные различия. Понимание этих отличительных черт поможет вам определиться в выборе наиболее удобного для конкретного типа информации приложения.

Одно из основных достоинств приложения Excel перед Access — относительная простота интерфейса пользователя. Несмотря на то, что Excel — довольно сложное и многогранное приложение, все его данные хранятся в одном месте — на листах рабочей книги. В противоположность этому, при открытии программы Access пользователь сразу же сталкивается с окном Базы данных и не всегда понимает, как и где нужно вводить данные. В базах данных Access существует множество типов объектов (таблицы, запросы, формы, отчеты и т.д.), в которых новичку разобраться совсем непросто.

В то же время Access имеет ряд достоинств, которыми не обладает программа Excel, и в этом вы сразу убеждаетесь, как только овладеваете со своим страхом и начинаете в нем работать. Здесь гораздо больше возможностей в структурной организации хранимой информации. К примеру, можно указать программе Access, что в некотором столбце таблицы вы собираетесь хранить только даты, и после этого система не позволит по ошибке ввести туда название заказчика. В той же ситуации Excel позволяет ввести какую угодно информацию в любую ячейку.

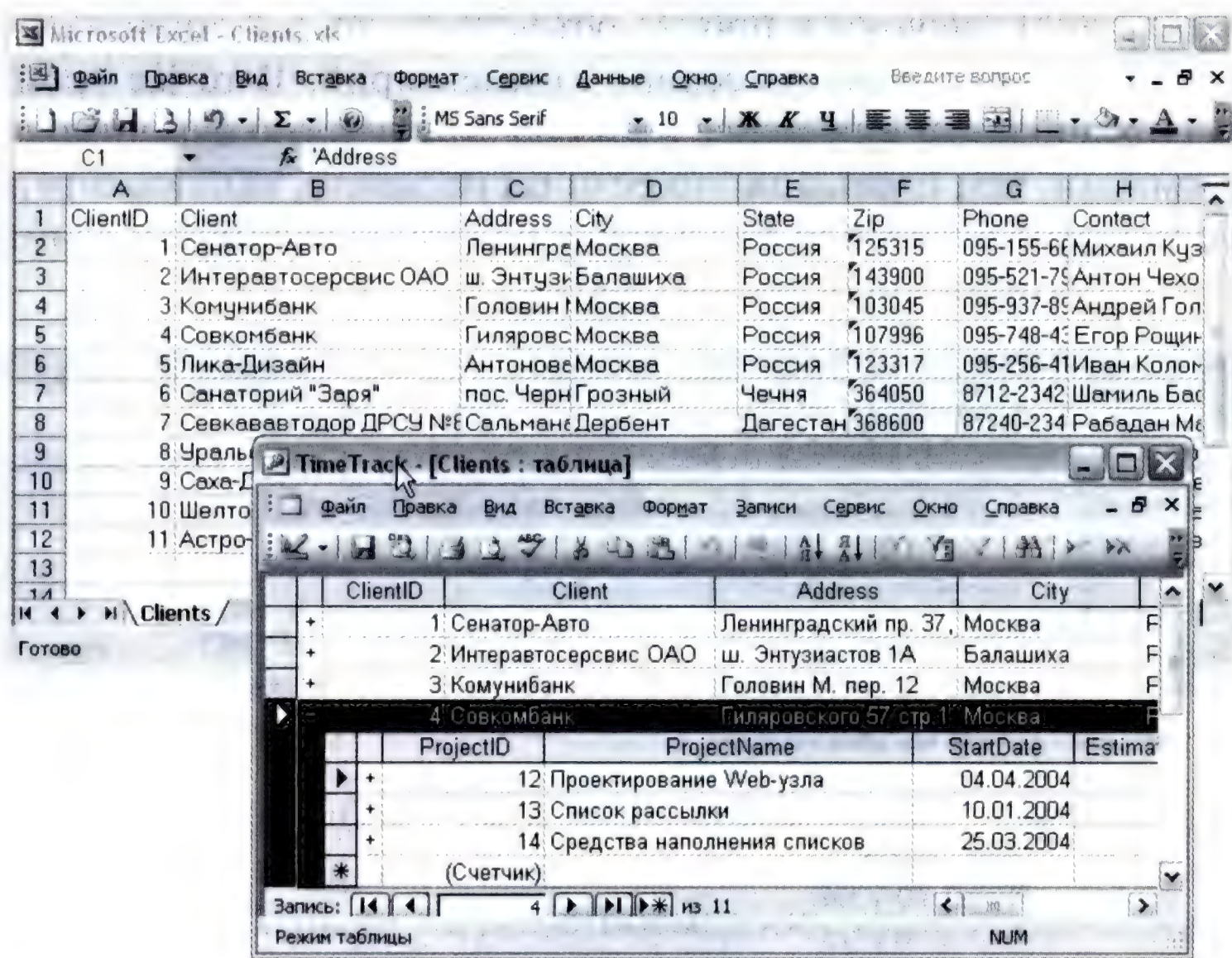


Рис. 1.1. Таблицы в Access и Excel

Приложение Access также учитывает концепцию *отношений* между данными. К примеру, можно установить связь между таблицами проектов и клиентов. При этом Access вырабатывает правила (в рассматриваемом примере его можно озвучить следующим образом: “Каждому проекту должен соответствовать некоторый заказчик”), позволяющие более гибко выполнять требуемые выборки и операции.

Примечание

На протяжении всей книги мы предполагаем, что вы уже знакомы с интерфейсом пользователя Access и основами работы с реляционными базами данных. Если это не так, рекомендуем вам прочесть книгу *Microsoft Access 2003. Самоучитель*, опубликованную издательством “Диалектика” (2004г.).

Access или OneNote?

Компания Microsoft в свой пакет Office 2003 включила еще одно приложение, предназначенное для хранения информации, — OneNote 2003 (рис. 1.2). Несмотря на то, что данная программа в основном была нацелена на пользователей карманных компьютеров, она является также и альтернативным решением для владельцев настольных компьютеров.

Программа OneNote создавалась для обеспечения гибкой записи информации любого типа в единую конструкцию. Вы можете либо набрать на клавиатуре, либо записать карандашом на планшете свои заметки, вставить или

перетянуть на лист рисунки и прочие объекты из других приложений и даже записать с микрофона свои голосовые комментарии. В то же время данная программа предполагает еще менее жесткую структурную организацию хранения информации, чем Excel. Так что если вы не знаете, как поместить информацию в структуры Access или Excel ввиду неорганизованности данных, подумайте об использовании программы OneNote.

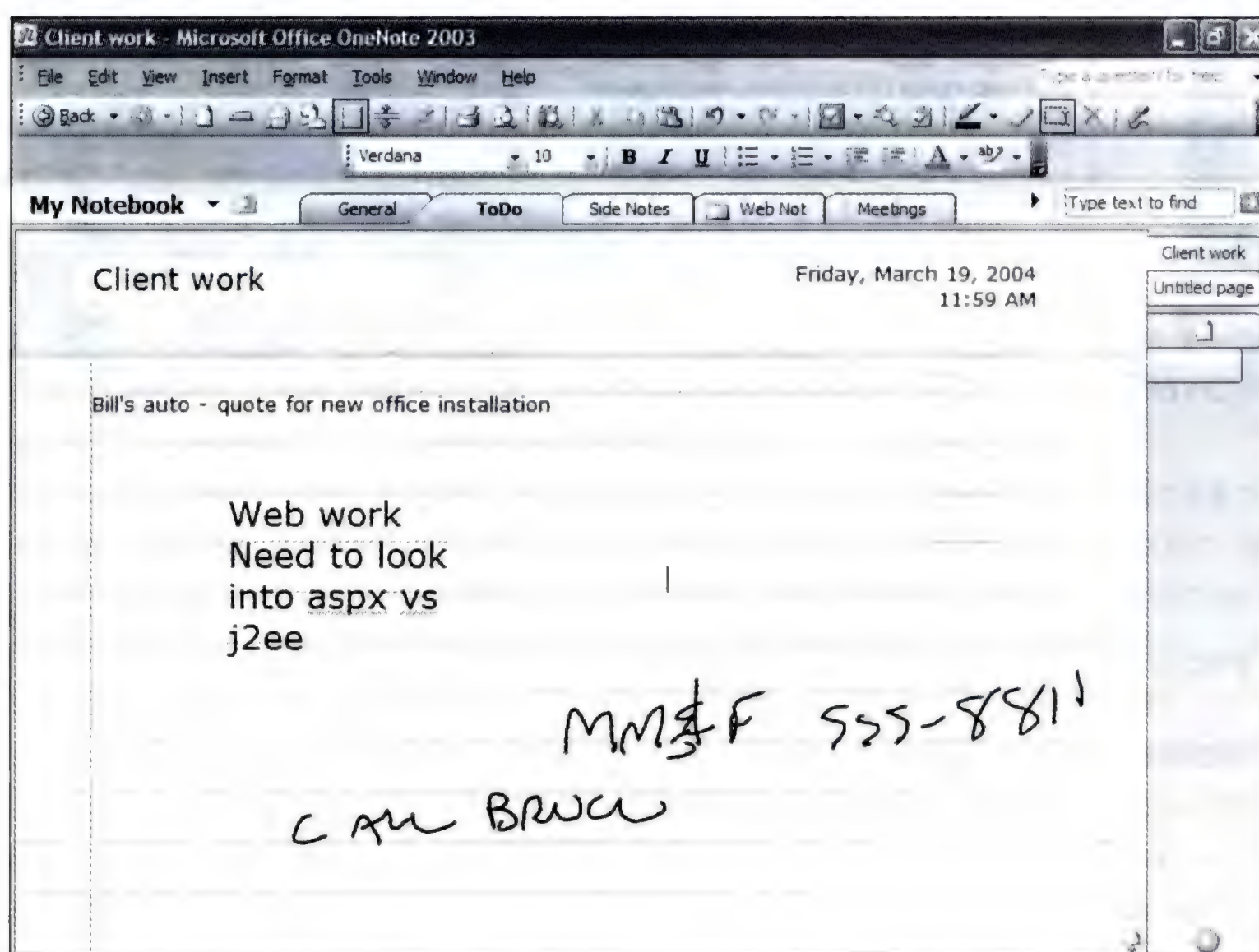


Рис. 1.2. Программа OneNote 2003 предлагает пользователю хранение информации в произвольном виде

Понятие программирования в Access

После того как ваш выбор в вопросе средств хранения информации пал на программу Access, нужно решить еще один важный вопрос: какой язык программирования использовать.

Язык VBA — не единственное средство автоматизации работы в Access. В зависимости от конкретных задач вы можете использовать как *макросы* приложения Access, так и язык структурированных запросов *SQL* (Structured Query Language).

Использование макросов

Вы, вероятно, уже знакомы с языком сценариев (макросов) приложения Access. Этот язык предлагает ограниченный набор средств автоматизации работы с базами данных. В качестве примера на рис. 1.3 показан элементарный макрос открытия формы, который можно как запускать не-

посредственно из окна Базы данных Access, так и подключить к кнопке какой-либо другой формы.

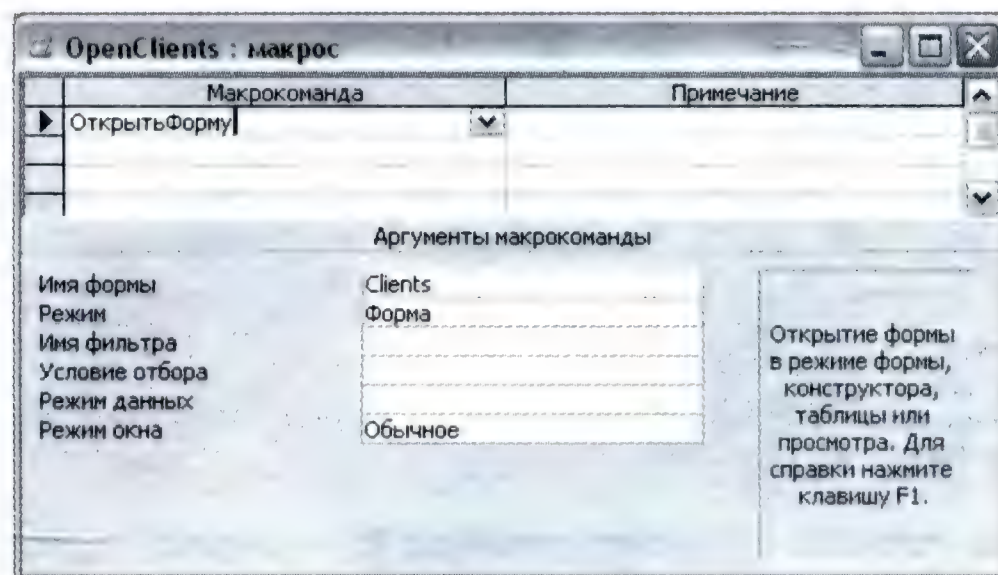


Рис. 1.3. Макросы приложения Access предлагают ограниченный набор средств автоматизации работы с базами данных

Макросы удобны в работе, но в то же время весьма ограничены. Естественно, существует немало баз данных, автоматизированных исключительно с помощью макросов, однако по мере повышения вашего профессионализма вы начнете видеть все недостатки этих средств. В частности, макросы имеют весьма узкий набор реакций на ошибки или другие состояния, отличающиеся от тривиальных.

Совет

Если вы решили в качестве основы автоматизации использовать язык VBA, то нет никакой необходимости отключать рабочие макросы, поскольку Access позволяет применять в одной базе данных разнотипные средства автоматизации.

Использование языка SQL

Вы наверняка слышали о языке структурированных запросов, известном под аббревиатурой *SQL*. Этот язык в приложении Access (а также и в других системах управления базами данных) используется для хранения запросов к базам данных. К примеру, с его помощью можно создать запрос, отбирающий из базы данных задач только те, которые связаны с проектированием Web-узлов. На рис. 1.4 показан пример запроса, выполняющего эту операцию.

Несмотря на то, что многие пользователи предпочитают работать с запросами в окне конструктора, программа Access все равно хранит запросы в виде инструкций SQL. Ниже показан запрос SQL, сформированный системой на основе представления в окне конструктора, показанного на рис. 1.4.

```
SELECT Projects.*
FROM Projects
WHERE (Projects.ProjectName Like "*Web*");
```

Несмотря на то, что SQL не является языком программирования широкого профиля, для работы в Access необходимо знать его основы. Так как большин-

ство задач автоматизации требуют извлечения из базы определенных данных, нужные запросы создаются с помощью языка SQL.

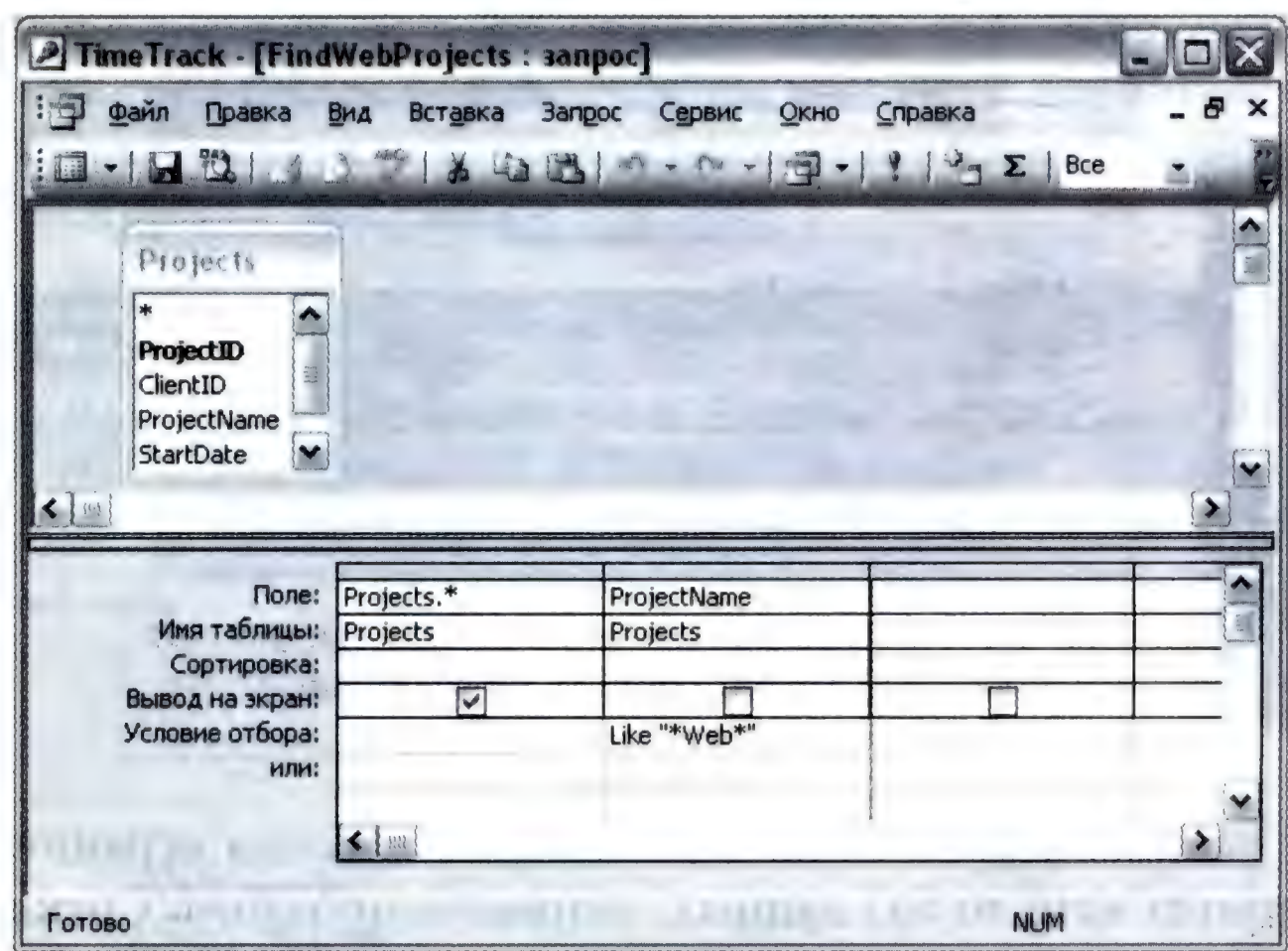


Рис. 1.4. Запрос, отбирающий проекты, которые связаны с проектированием Web-узлов, и открытый в окне конструктора Access

➔ Если вы не знакомы с языком SQL, то можете в качестве справочника использовать приложение “Обзор языка SQL” данной книги.

Использование языка VBA

В заключение остановимся на языке VBA — центральном объекте рассмотрения в настоящей книге. Этот язык позволяет автоматизировать практически любую операцию, доступную в приложении и базах данных Access. Вот список операций (они описаны в книге), который позволяет выполнить язык VBA:

- автоматизация ввода данных в формы Access;
- добавление новых элементов в списки;
- компоновка данных, помещаемых в отчеты;
- работа с данными без открытия форм;
- автоматизация других приложений из Access;
- импорт и экспорт данных из файлов XML.

Все эти операции являются лишь небольшим срезом практически безграничных возможностей языка VBA. Чтобы продемонстрировать на примере доступные в этом языке средства, начнем с построения простой базы данных, которая постепенно, с каждым новым практикумом, будет усложняться и становиться все более удобной и полезной.

ПРАКТИКУМ

Использование базы данных практикумов

В процессе работы над материалом книги начинайте постепенно применять изученные методы автоматизации в собственных базах данных. Для иллюстрации теоретического материала в книге используется прилагающаяся в качестве демонстрационной база данных под названием TimeTrack, которую вы можете загрузить с Web-узла настоящей книги. Пока будем считать, что в этой базе данных нет никакого программного кода; в ней содержится только информация о проектах и выполненных работах в некоторой гипотетической программистской компании.

Когда вы откроете базу данных примеров, вы увидите форму Switchboard, показанную на рис. 1.5. Эта форма служит интерфейсом к остальным четырем формам и одному отчету, составляющим приложение.

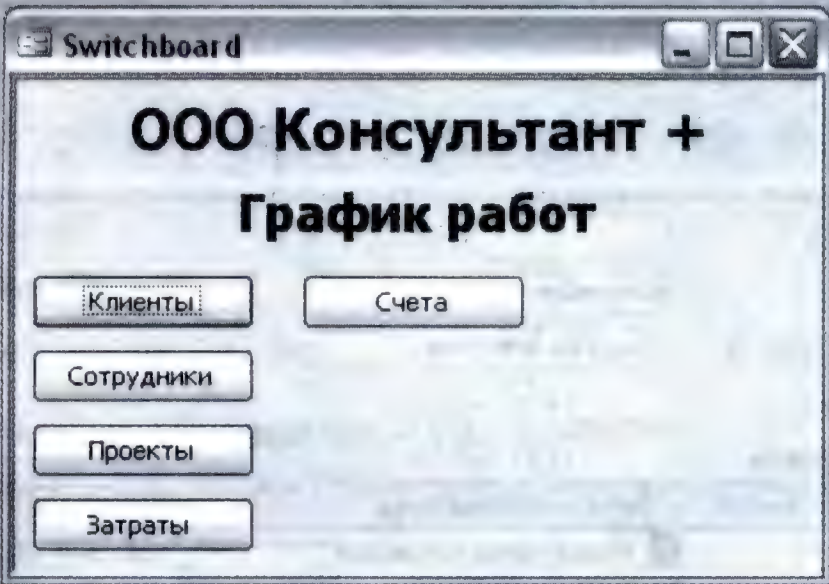


Рис. 1.5. Форма Switchboard является отправной точкой в приложении TimeTrack

Щелкнув на кнопке Клиенты, вы увидите список клиентов наряду с исполненными по их заказам проектами (рис. 1.6).

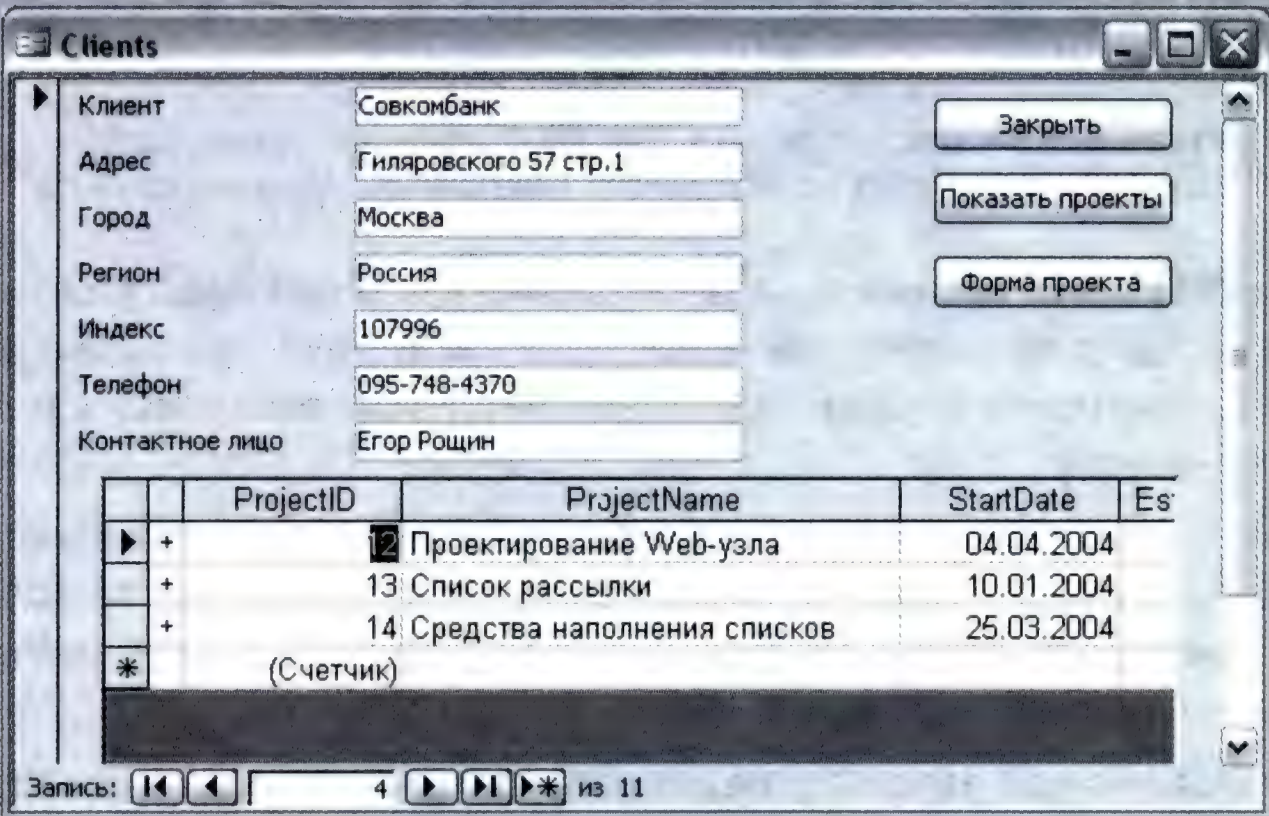


Рис. 1.6. В единой форме вы можете работать как с реквизитами клиентов, так и с информацией о выполненных для них проектах

На рис. 1.7 показана форма Employees, в которой вводится информация о сотрудниках компании, назначенных для работы над конкретными задачами.

Employees - форма

Табельный №: 3

Имя: Руслан

Фамилия: Пилипейко

Запись: 3 из 14

Рис. 1.7. Форма Employees позволяет вводить информацию о сотрудниках

Форма Projects (рис. 1.8) является центральной при работе с информацией о проектах. Для каждого из проектов из списка выбирается заказчик. Также для каждого из проектов можно ввести список задач и расценки, ассоциируемые с каждой задачей.

Projects

№ проекта: 11

Клиент: Комунибанк

Название проекта: Создание Web-узла

Дата начала: 19.06.2004

Дата окончания: 11.11.2004

TaskID	TaskName	HourlyRate
53	Управление проектом	150,00р.
54	Разработка	105,00р.
55	Тестирование	65,00р.
56	Сопровождение	45,00р.
* (Счетчик)		

Запись: 1 из 4

Запись: 10 из 30

Рис. 1.8. Каждый проект состоит из отдельных задач, для каждой из которых назначается конкретный уровень оплаты

Последняя форма приложения показана на рис. 1.9. В ней вводится информация о затраченном на каждую задачу времени, дате выполнения, а также о ее исполнителе. Все это гарантирует корректное предоставление заказчику счетов на оплату предоставленных услуг.

В заключение, на рис. 1.10 показана счет-фактура, выставляемая клиенту на основе выполненных за определенный период работ. При запуске на выполнение отчета приложение требует выбора заказчика, а также ввода дат, ограничивающих период, за который выставляется счет.

В настоящий момент вся автоматизация базы данных выполнена с помощью макросов. Однако мы это так не оставим!

Timeslips

Сотрудник:

Задача:

Дата выполнения работ:

Затрачено часов:

Запись: из 6925

Рис. 1.9. Информация о затраченном на задачу времени и ее исполнителе

TimeTrack [BillingReport]

Файл Правка Вид Сервис Окно Справка

Счет-фактура

Комунибанк

Головин М. пер. 12

Москва, Россия 103045

Кому: Андрей Головин

Название проекта: Создание Web-узла

Название задачи	Разработка	Ставка за чел/час:	
			105,00р.
Дата работ	Кол-во часов	Итого по строке	
19.06.2004	2,25	236,25р.	
19.06.2004	12,5	1 312,50р.	
23.06.2004	3	315,00р.	

Страница: 14 1

Готово NUM

Рис. 1.10. Отчет BillingReport вычисляет сумму, выставляемую клиенту за работы, проведенные в течение заданного периода времени

Редактор Visual Basic

2

Первый взгляд на редактор Visual Basic

Редактор Visual Basic (*VBE*) является тем интерфейсом, который используют для написания программ на языке VBA. На рис. 2.1 показано главное окно этого редактора, в котором открыта база данных Time-Track. Простейшим способом вызова редактора VBE является открытие в Access базы данных и нажатие клавиш <Att+F11>. (Если вы будете открывать базу данных, прилагаемую к настоящей главе, то при вызове редактора Visual Basic откроется фрагмент текста программы.)

В этой главе будет представлено множество компонентов, с которыми придется работать в редакторе VBE при вводе и редактировании программ. Мы не будем тратить много времени на подробное описание каждого из элементов, инструментов и команд меню — вы познакомитесь с ними при работе с примерами программ. Сейчас же мы просто познакомимся с общей средой разработки, чтобы в дальнейшем работа в ней не вызывала у вас особых сложностей.

По умолчанию редактор VBE отображает следующие компоненты:

- меню, принятое по умолчанию;
- стандартную панель инструментов;

В ЭТОЙ ГЛАВЕ

Первый взгляд на редактор Visual Basic.....	35
Введение в модули VBA.....	38
Ввод и запуск программ VBA	40
Получение справки по программированию.....	43
Выработка навыков по вводу кода	45

- панель проекта с иерархическим списком элементов, содержащихся в текущей базе данных и имеющихся в ней в качестве ссылок;
- окно свойств с параметрами объектов;
- окно Immediate, позволяющее посмотреть на результат выполнения программы и отдельных команд.

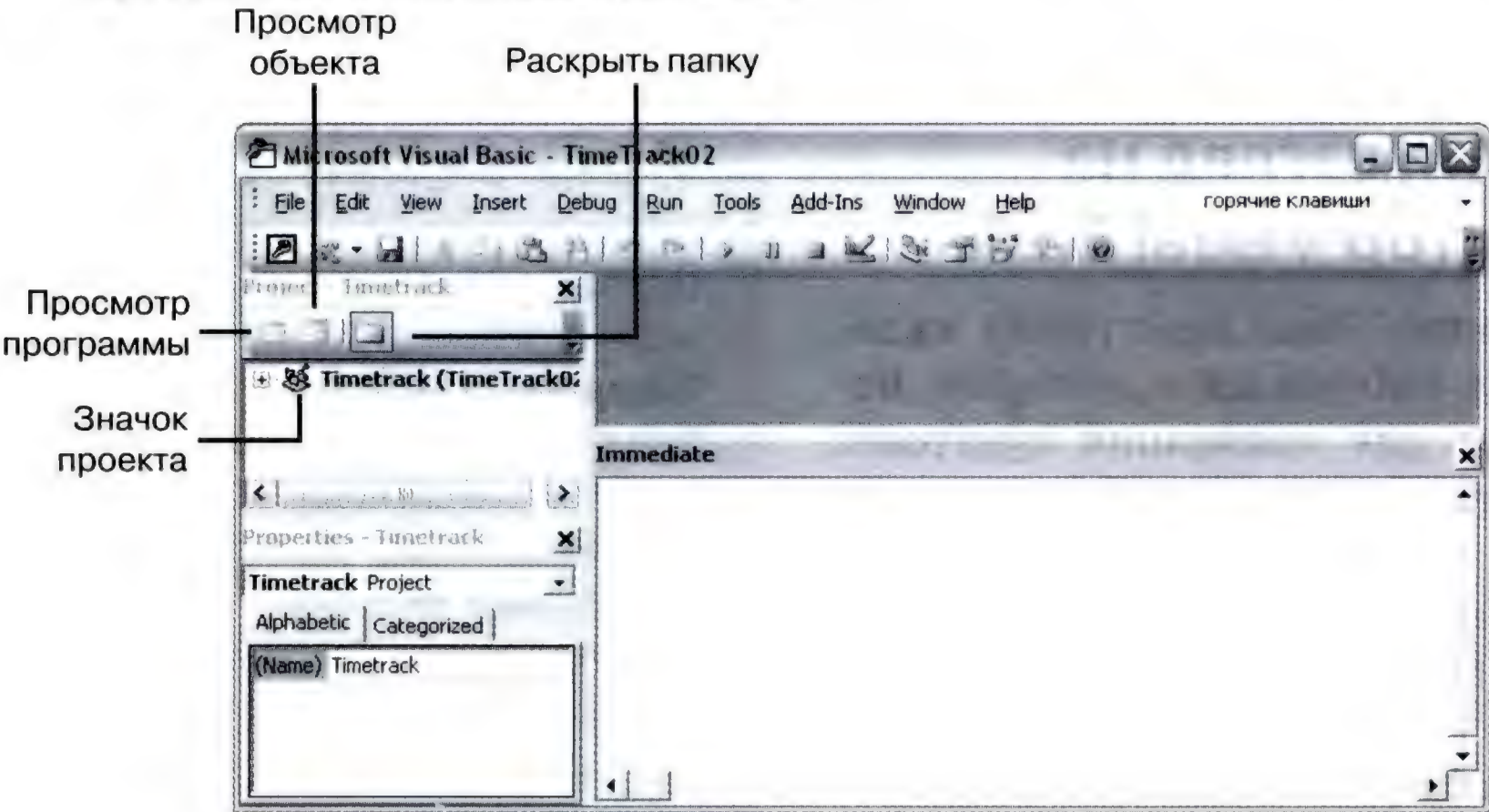


Рис. 2.1. Добро пожаловать в редактор Visual Basic!

Совет

Отдельные окна редактора VBA можно закрепить на границах главного окна программы, равно как панели инструментов и меню. Для переключения между закрепленным и плавающим режимом окон достаточно дважды щелкнуть на их заголовке.

В табл. 2.1 перечислены средства стандартной панели инструментов редактора VBE, которая показана на рис. 2.1, а также соответствующие им горячие клавиши и команды меню. Этими инструментами вы будете пользоваться практически постоянно.

Таблица 2.1. Стандартная панель инструментов

Название команды	Назначение	Горячие клавиши	Команда меню
View Microsoft Office Access	Отображение окна Access 2003, без закрытия окна VBE	Alt+F11	View⇒Microsoft Office Access
Insert Module	Создание нового пустого модуля. В списке выбирается тип модуля		Insert⇒Module
Find	Поиск в модуле заданного слова или фразы	Ctrl+F	Edit⇒Find

Окончание табл. 2.1

Название команды	Назначение	Горячие клавиши	Команда меню
Undo	Отмена последней операции или команды (если такое возможно)	Ctrl+Z	Edit⇒Undo
Redo	Восстановление последней отмененной операции или команды		Edit⇒Redo
Run Sub/Userform	Выполнение текущей открытой процедуры или продолжение ее выполнения после останова	F5	Run⇒Run Sub/UserForm
Break	Прерывание выполнения процедуры	Ctrl+Break	Run⇒Break
Reset	Завершение выполнения процедуры с восстановлением исходных значений переменных	Shift+F5	Run⇒Reset
Design Mode	Переключение формы в режим конструктора		Run⇒Design Mode
Project Explorer	Открытие панели проекта	Ctrl+R	View⇒Project Explorer
Properties Windows	Открытие окна свойств объекта	F4	View⇒Properties Windows
Object Browser	Открытие панели объектов	F2	View⇒Object Browser
Toolbox	Открытие панели элементов		View⇒Toolbox
VBA Help	Вызов справки	F1	Help⇒Microsoft Visual Basic Help

Все остальные средства доступны на трех дополнительных панелях инструментов, две из которых показаны на рис. 2.2 и 2.3. С использованием этих инструментов вы познакомитесь ближе в ходе рассмотрения примеров. Четвертая панель инструментов — UserForm — также доступна в программе, однако в рамках настоящей книги мы не будем соприкасаться с темой использования форм пользователя, а равно и с этой панелью инструментов.

Для отображения какой-либо из панелей инструментов нужно выбрать в меню команду View⇒Toolbars и щелкнуть на названии соответствующей панели инструментов — Debug (Отладка), Edit (Редактирование) или Standard (стандартная). Для выключения любой из панелей нужно в том же меню снять отметку с соответствующей строки.

Совет

Средства *UserForms* редактора VBE мы не будем рассматривать по той причине, что программа Access 2003 имеет собственные, более мощные средства разработки форм. А чем же вызвано включение в редактор VBE средств UserForms? Ответ прост: как редактор VBE, так и сам язык VBA являются компонентами общего использования. Реализации и приложения VBA вы найдете в десятках разных пакетов, в том числе в офисных программах Microsoft Office, инженерном графическом пакете AutoCAD, пакете Peachtree Office Accounting и многих других. Среди них есть и такие, которые не имеют собственных средств разработки форм. Именно по этой причине средства UserForms включены в редактор VBE.



Рис. 2.2. Инструменты отладки помогут вам быстро отыскать ошибки в тексте программ



Рис. 2.3. Инструменты редактирования используются для создания эффективных программ

Введение в модули VBA

После запуска редактора VBE вы, вероятно, захотите просмотреть существующую программу или ввести новую. Все программы на языке VBA содержатся в одном из трех типов модулей:

- **Стандартные модули.** Они содержат фрагменты программы, которые не связаны с каким-то конкретным объектом.
- **Модули объектов.** Содержат фрагменты программы, которые соответствуют прикрепленной форме или отчету. Если вы уже написали процедуру обработки какого-либо события, то определенно сделали это в

модуле объекта. Каждая форма и отчет в базе данных Access имеет связанный с собой модуль объекта.

- **Модули класса.** Они содержат фрагменты программ, определяющие объекты пользователя. (Более подробно модули классов описаны в главе 8.)

Для вставки стандартного модуля или модуля класса нужно выбрать в меню команду, соответственно, **Insert⇒Module** или **Insert⇒Class Module**. Система Access вставит в базу данных модуль соответствующего типа.

На рис. 2.4 показан стандартный модуль, открытый в редакторе VBE.1. Обратите внимание, как вставка нового модуля изменяет содержание окон проекта и свойств объекта. В открывшееся окно стандартного модуля можно ввести текст программы, независимый от какого-то конкретного объекта или события. В этом тексте вы можете использовать любые объекты и события, однако данный модуль не будет определен ни в одном объекте или предопределенном событии.

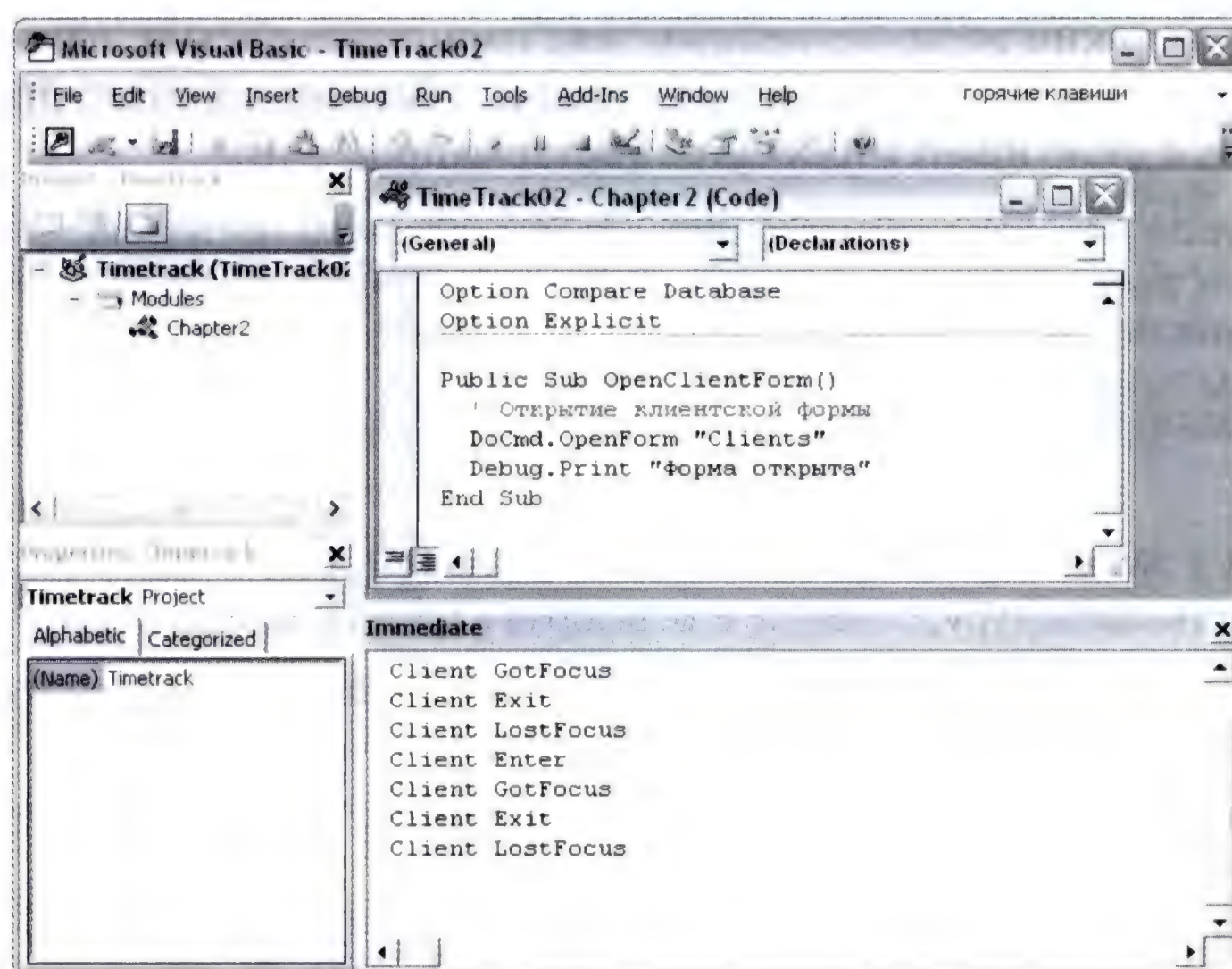


Рис. 2.4. Модули содержат программы на языке VBA, автоматизирующие базу данных

Модули объектов связаны с созданными формами или отчетами. На самом деле они являются модулями классов, но вам нет никакой необходимости полностью раскручивать весь клубок, если требуется всего лишь добавить какой-либо фрагмент программного кода к какому-то элементу. Программа в модуле объекта, как правило, связана с обработкой какого-либо события, связанного с этим объектом. В то же время, модули объектов могут содержать и процедуры, не связанные ни с каким событием.

Совет

Существует множество горячих клавиш, связанных с работой в редакторе VBE. Часть из них уже была перечислена в табл. 2.1, однако существует и множество других. К примеру, при нажатии <Shift+F10> отображается контекстное меню активного окна (идентичное меню отображается при щелчке правой кнопкой на свободной области любого окна). Если вы хотите более подробно узнать о горячих клавишах, используемых в редакторе VBE, откройте раздел “Keyboard Shortcuts” в справочной системе программы.

Ввод и запуск программ на языке VBA

Ввод программ в редакторе VBE ничем не отличается от ввода простого текста в обычном текстовом процессоре — в окне модуля вам нужно набрать необходимые операторы, переменные и выражения.

Как только вы ближе познакомитесь с языком VBA, ввод текста программ станет для вас таким же естественным, как написание письма, однако до этого времени вам потребуется небольшая помощь. Именно для этого предназначено диалоговое окно *Insert Procedures* (Вставка процедур).

→ *Процедурой* называют самодостаточную исполняемую последовательность программных операторов и выражений. В разделе “Типы процедур” главы 4 эта тема освещается более подробно.

Для создания новой процедуры вам потребуется выполнить следующие действия:

1. Если у вас еще не открыт ни один модуль (см. рис. 2.4), в который можно вводить процедуру, создайте его, выбрав в меню команду *Insert⇒Module*.
 2. Для открытия окна вставки процедуры выберите в меню команду *Insert⇒Procedure*.
 3. Введите короткое, но описательное название процедуры. В данном случае назовем процедуру *OpenClientForm*.
 4. Установите переключатели в разделах *Type* (Тип) и *Scope* (Область определения) в соответствующее положение. При создании статических переменных нужно установить флажок в поле *All local variables as Statics* (Все локальные переменные статичны). В нашем примере установим тип как процедуру (*Sub*), определим переменные как общедоступные (*Public*) и не статичные (снимем флажок статичности).
 5. Щелкните на кнопке *OK*.
- Область определения переменных более подробно рассматривается в разделе “Объявление общих и частных процедур” главы 4.

После того как вы щелкните на кнопке *OK*, редактор VBE отобразит то, что называют *скелетом процедуры* (рис. 2.6). В скелете программный код опреде-

ляется либо как функция, либо как процедура, задается его имя, и весь фрагмент закрывается оператором End. В скелете процедуры не содержится какой-либо исполняемый код — его вводом вам придется заняться самостоятельно.

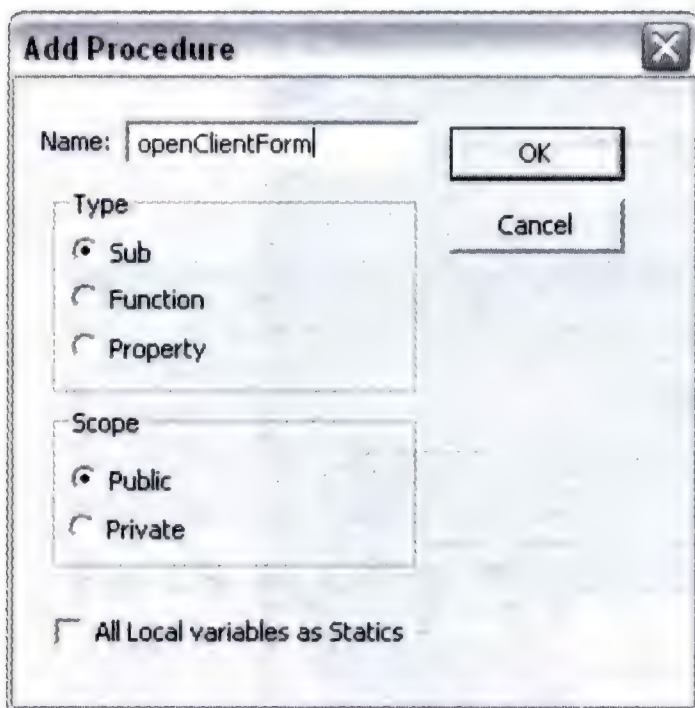


Рис. 2.5. Для определения новой процедуры установите эти параметры

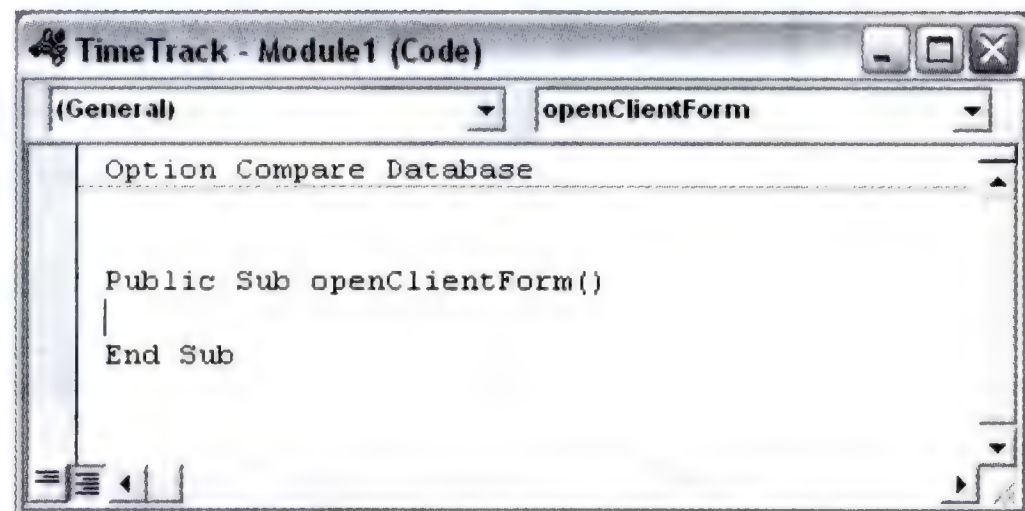


Рис. 2.6. Редактор VBE создал скелет процедуры

Будете ли вы начинать со скелета, созданного редактором VBE, или будете создавать его вручную, вашей основной задачей является ввод текста самой программы. Именно этому вы будете учиться на протяжении всей этой книги. Теперь же, для того чтобы продемонстрировать процедуру работы с редактором, введем в скелет процедуры пару строк исполняемого текста:

```
Public Sub OpenClientForm()
    'Открытие клиентской формы
    DoCmd.OpenForm "Clients"
    Debug.Print "Форма открыта"
End Sub
```

Если в программе пока не открыто окно Immediate, нажмите комбинацию клавиш <Ctrl+G>. Теперь установите указатель мыши в любом месте внутри написанной процедуры и нажмите клавишу <F5>. Во-первых, в окне Access откроется форма Clients (рис. 2.7), как будто вы ее открыли вручную (на самом деле в текущий момент эту форму вы можете не увидеть; для этого вам потребуется переключиться в окно программы Access, если вы того захотите). Во-вторых, в окне Immediate программа выведет сообщение, заданное в программе (рис. 2.8).

Обычно в окне Immediate не выводят каких-либо значений — все значения используются в коде программы. Сейчас же достаточно того, что в этом окне мы увидели результат выполнения программы — аналогичным образом это окно можно использовать и для отладки программ.

Сохранение программ

Для сохранения нового модуля или изменений, выполненных в существующем модуле, щелкните на кнопке Save (Сохранить) стандартной панели инструментов

или выберите в меню команду **File⇒Save имя_проекта**. При сохранении нового модуля программа Access 2003 открывает диалоговое окно **Save as** (Сохранить как). Вы можете принять имя, предложенное программой Access по умолчанию, или ввести собственное (рис. 2.9), после чего щелкнуть на кнопке **OK**.

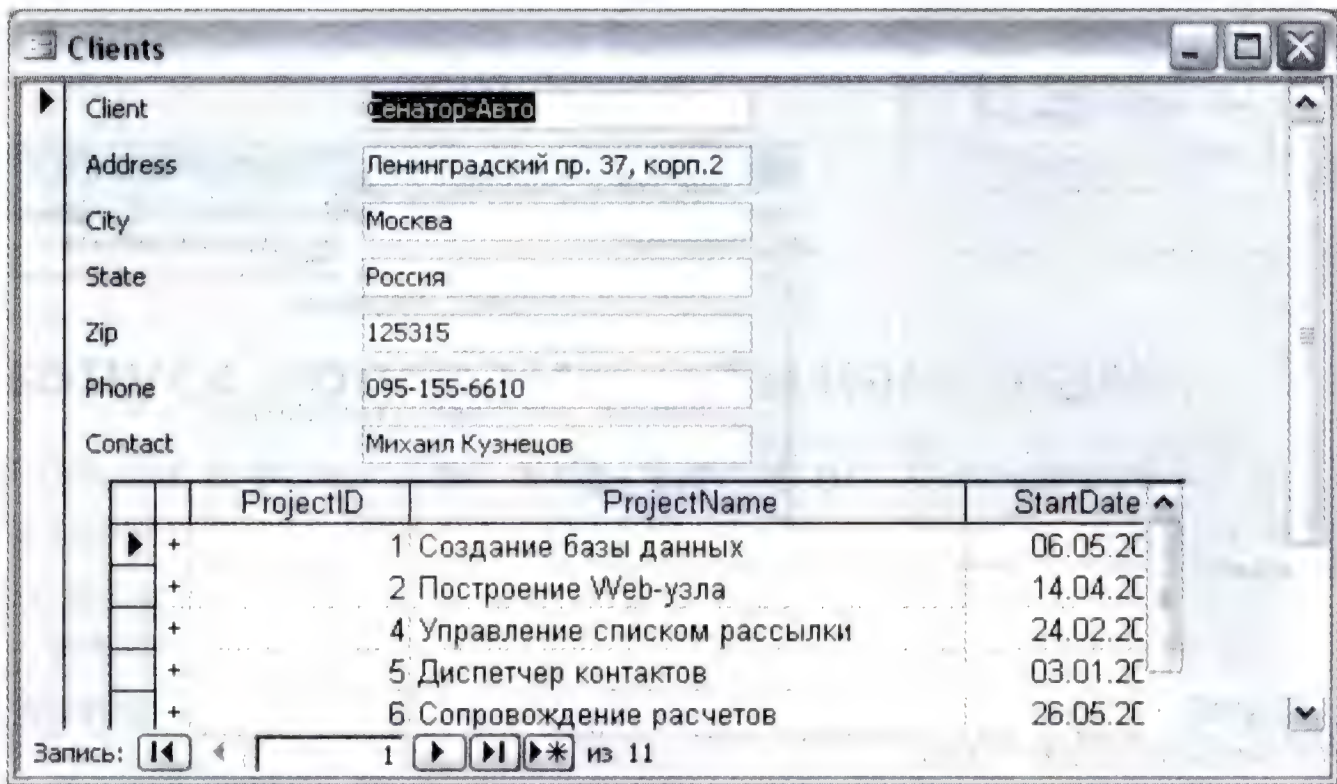


Рис. 2.7. При запуске процедуры на выполнение открывается форма Access

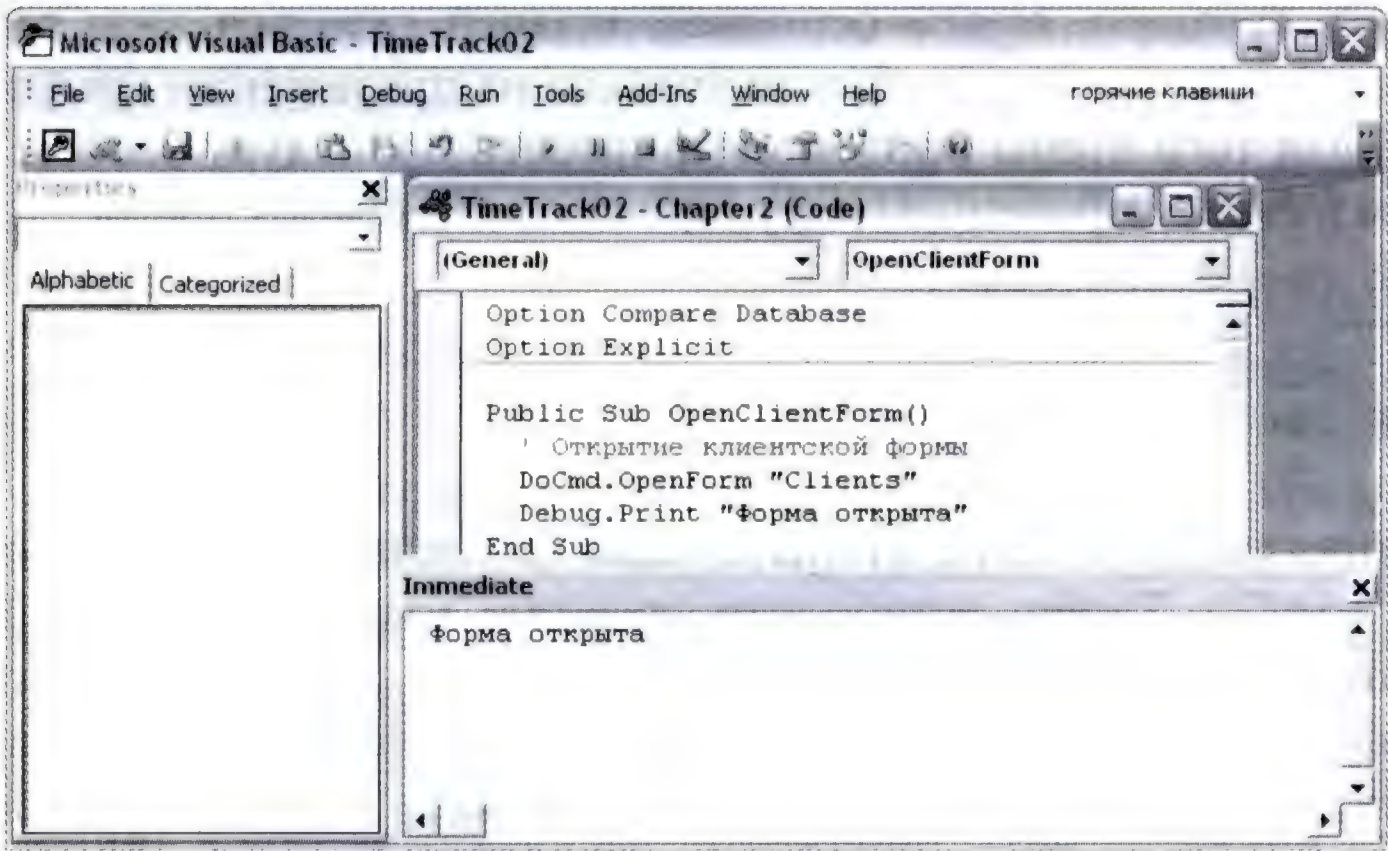


Рис. 2.8. В результате выполнения процедуры в окне Immediate отображается сообщение

Предупреждение

Можно закрыть редактор VBE без потери несохраненного кода, хотя это и не рекомендуется. В то же время, как только вы закроете модуль, несохраненный код будет безвозвратно утерян.

Обратите внимание, что новый модуль уже перечислен в списке панели проекта (рис. 2.10) (здесь он указан под именем *NavigationProcedures*).

Также стоит обратить внимание на то, что выделение нового модуля на панели проекта приводит к изменению содержимого окна свойств объекта. В настоящее время в этом окне не отображаются никаких свойств кроме имени модуля. Впоследствии вы ближе познакомитесь с зависимостью этих окон друг от друга.

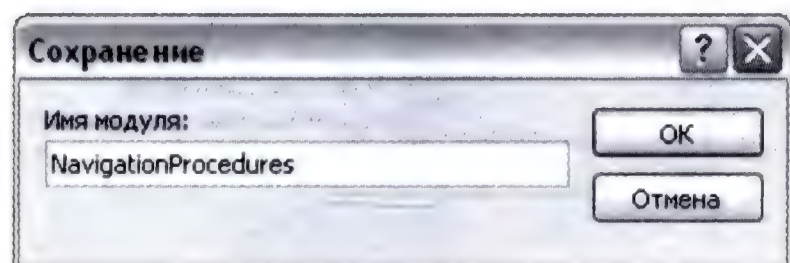


Рис. 2.9. Не забудьте сохранить новый модуль

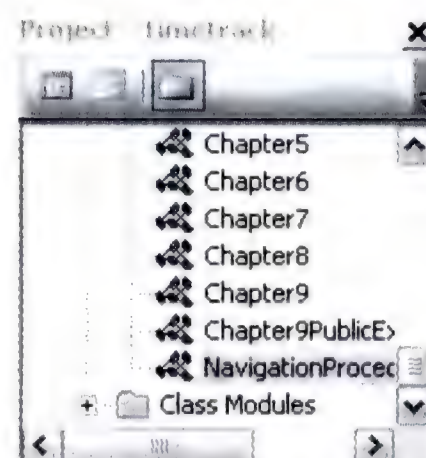


Рис. 2.10. При сохранении модуля его имя отображается на панели проекта

Получение справки по программированию

Только немногие программисты способны с первого раза безошибочно написать программу, большинство из нас основное время тратит на исправление ошибок. К счастью, редактор VBE оснащен мощной и удобной справочной системой, которая полностью обособлена от справки приложения Access 2003.

Откройте справочную панель редактора VBE, показанную на рис. 2.11, и вы получите доступ к этой информации. Это можно сделать двумя способами: выбрать в меню команду **Help**⇒**Microsoft Visual Basic Help** или команду **View**⇒**Toolbars**⇒**Область задач**.

Панель справки в редакторе VBE работает аналогично справочной системе старых версий, однако она не унаследовала окно оглавления. Введите в поле **Поиск** слово или фразу и, нажав клавишу <Enter>, в окне результатов поиска вы получите список тем справки, близких к введенным ключевым словам.

Совет

Самым простым способом получения справки является позиционирование курсора в центре ключевого слова (функции, метода и т.п.) и нажатие клавиши <F1>. После этого система отобразит в отдельном окне справку по связанному с данным словом вопросу.

Полезные горячие клавиши при вводе текста программы

Вы можете полагаться на информацию о языке VBA, содержащуюся в справочной системе редактора VBE, однако иногда такая полная информация и не требуется. По мере ввода программного кода в модуль включается специальная интеллектуальная функция, называемая *IntelliSense*. Когда редактор VBE считает, что догадался, что именно вы собираетесь ввести, он открывает окно, содержащее варианты завершения вводимого оператора.

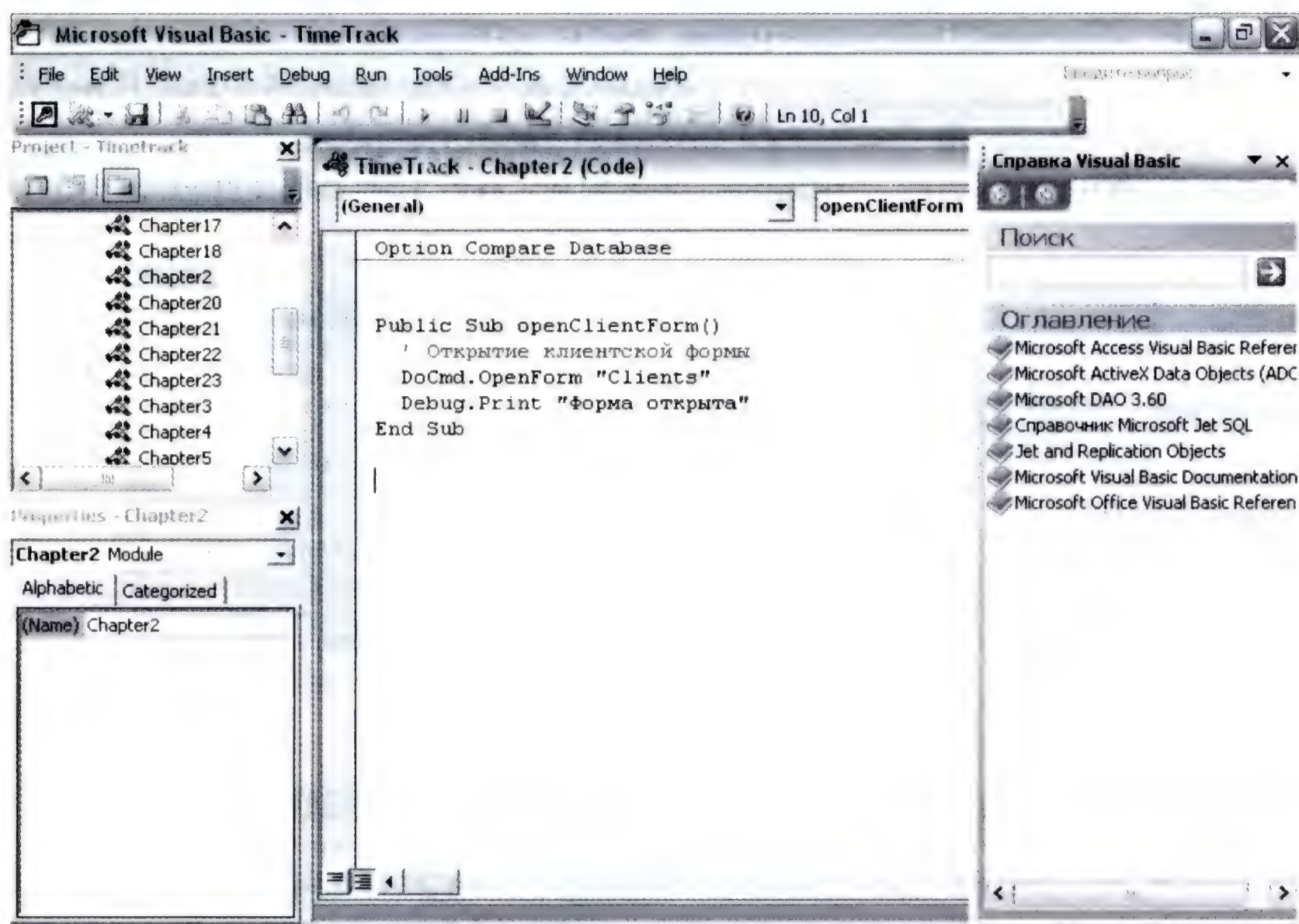


Рис. 2.11. На новой панели поиска редактора VBE можно отыскать справку по интересующему вас вопросу

Этот отображаемый список всегда зависит от контекста. Вы это заметите, как только попытаетесь ввести первые символы программы. Для того чтобы проиллюстрировать работу этой функции, давайте поработаем со следующим примером.

1. В свободной строке окна **Immediate** введите слово **DoCmd.** (включая символ точки). Сразу после ввода точки откроется контекстное меню, показанное на рис. 2.12.
2. Выберите в списке функцию **OpenForm**. Для этого вы можете воспользоваться полосой прокрутки или многократным нажатием клавиши стрелки вниз. Дважды щелкните на строке **OpenForm** или выделите ее и нажмите клавишу <Tab>.
3. Введите знак пробела и редактор откроет новый список. На этот раз он будет содержать аргументы функции **OpenForm** (рис. 2.13).
4. Первым аргументом является имя формы. После ввода соответствующей ссылки и запятой, в списке отобразятся константы для следующего аргумента (рис. 2.14). Об аргументах и константах, используемых методом **OpenForm**, вы узнаете в последующих главах книги.

Если вам нужна помощь, а список не открывается, щелкните правой кнопкой на ключевом слове, и откроется список, показанный на рис. 2.15. Щелкните на нужном пункте и вы получите соответствующую справку. (Это дейст-

вие в окне Immediate менее эффективно, так как результирующий список оказывается намного короче.)

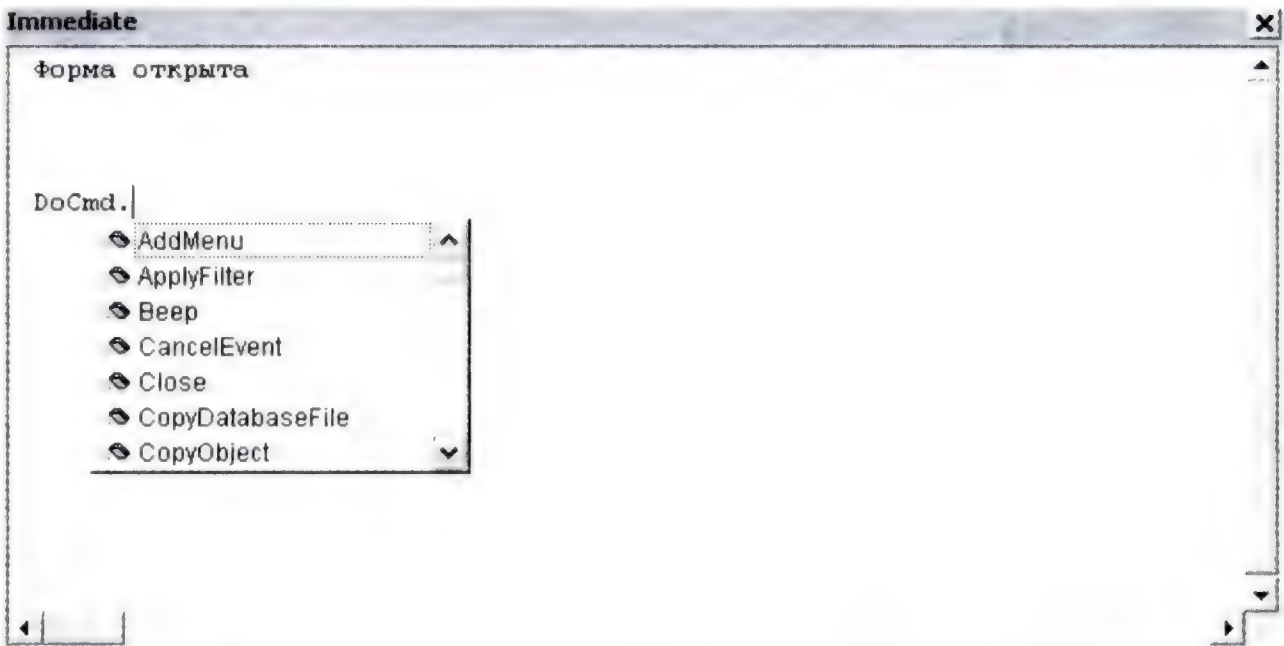


Рис. 2.12. Раскрывающийся список ограничивает выбор теми значениями, которые уместны для уже введенного слова

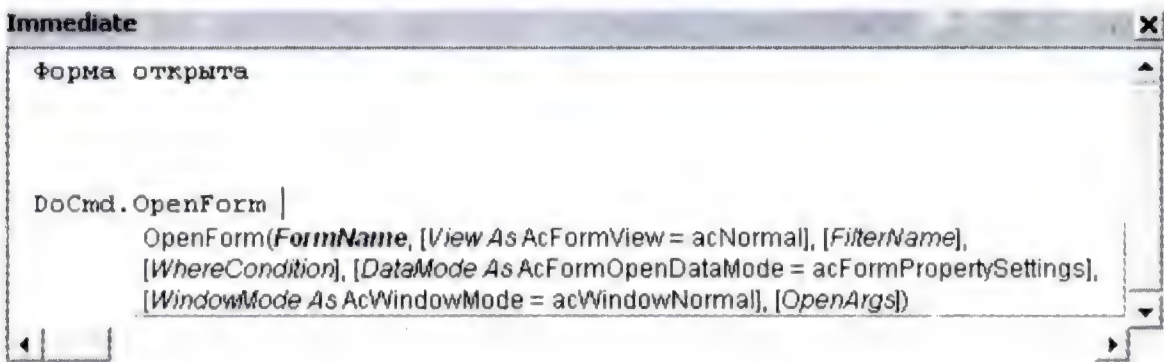


Рис. 2.13. Теперь в списке содержатся аргументы метода

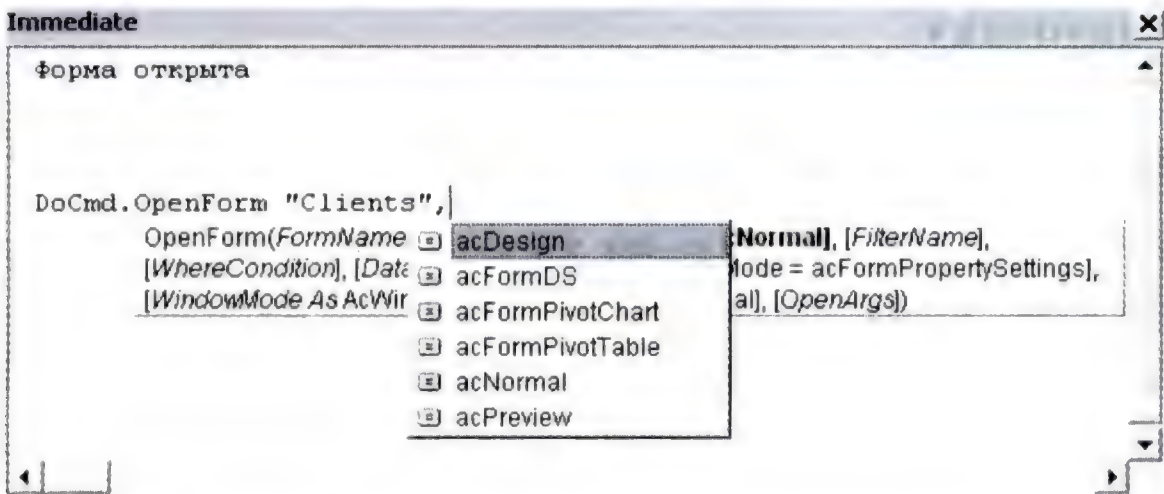


Рис. 2.14. Выберите из списка нужную константу

Выработка навыков по вводу кода

При настройке базы данных с помощью кода VBA редко кто ставит своей целью исключительно выполнение поставленной задачи. С самого начала вы должны думать как профессиональный программист и писать такой текст программы, который был бы понятен другому человеку, и логику которого было бы как можно проще отследить. Прислушайтесь к некоторым рекомендациям.

- При задании имен объектов и переменных будьте последовательны и придерживайтесь некоторых единых правил.

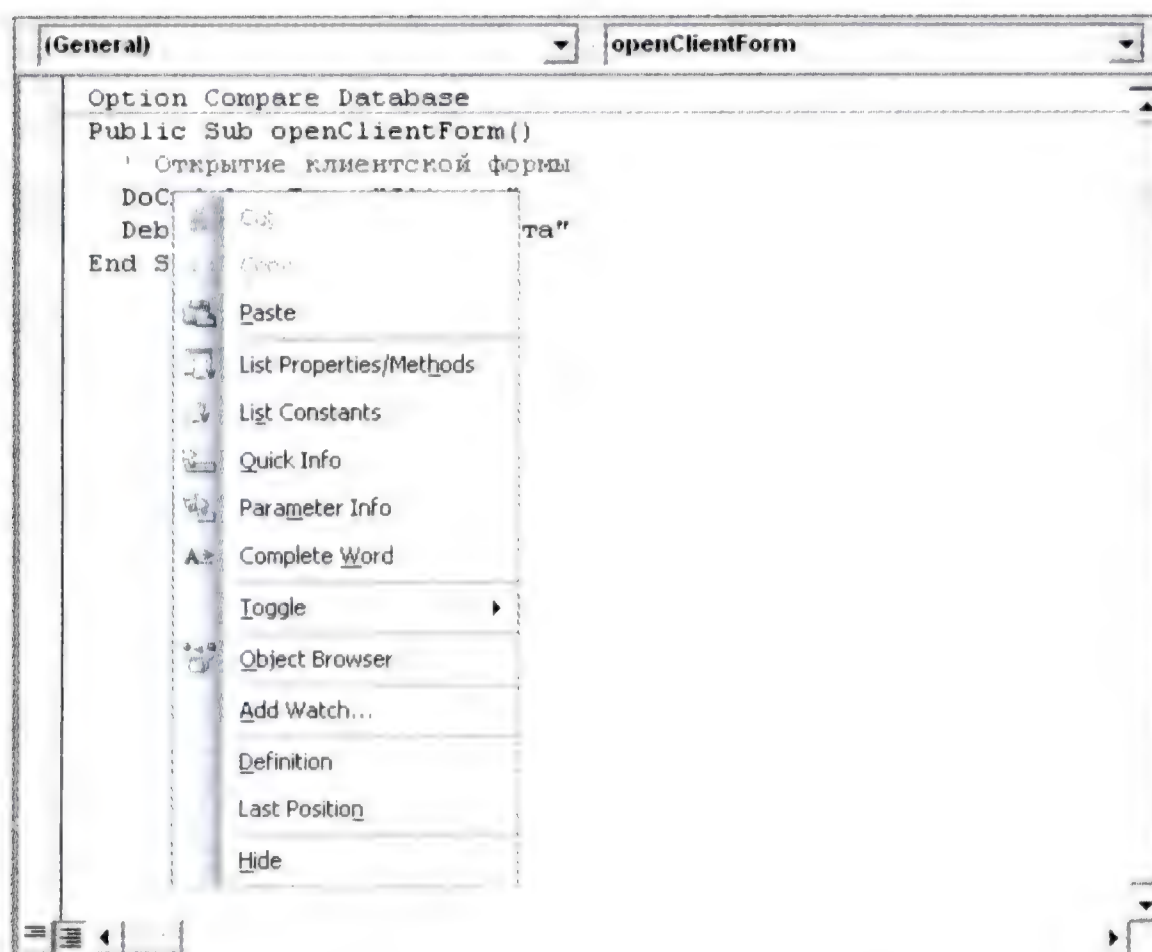


Рис. 2.15. С помощью щелчка правой кнопкой на ключевом слове вы можете получить дополнительную справку

- Устанавливайте в программе отступы, чтобы легче было разобраться в ее структуре.
- Комментируйте программу, чтобы впоследствии вспомнить, что именно выполняется в конкретном фрагменте кода. Кроме того, вполне возможно, что редактировать текст программы придется уже другому человеку.

Использование соглашений об именах

По мере написания программы на языке VBA вам придется создавать переменные и объекты, которые требуют присвоения имени. Вы можете задавать любые имена, однако при этом рекомендуется придерживаться ряда общепринятых соглашений.

- ➔ О создании и использовании переменных вы узнаете в разделе “Объявление переменных и констант” главы 3, а о создании объектов — в главе 8.

Соглашением об именах называют набор правил, в которых описывается структура имен переменных и объектов. Ваша организация может иметь собственные правила именования объектов, и если это так, вы обязаны их придерживаться. Если же таких правил не существует, все равно выработайте соглашения, упорядочивающие имена в программе.

Одним из хороших правил является использование в именах префиксов, указывающих на тип переменной или объекта. При этом можно использовать следующий синтаксис:

классИмя_объекта

где *класс* — трехсимвольный префикс, указывающий на тип объекта или переменной, а *Имя_объекта* — описательное имя. Обратите внимание на регистр символов в этой конструкции:

- *класс* — здесь используются символы из нижнего регистра;
- *Имя_объекта* — здесь первая буква каждого слова выделяется верхним регистром.

На протяжении всей книги будет использовано общепринятое, так называемое *Венгерское соглашение*. Это соглашение следует описанному выше правилу. В дополнение к нему мы будем использовать так называемую *естественную систему имен*, в которой в имени закладывается назначение объекта или тип используемых им данных.

В табл. 2.2 и 2.3 перечислены наиболее часто используемые префиксы. Ни один из этих списков не является полным, однако все эти префиксы вам, скорее всего, придется применять на практике, именно они будут использованы в настоящей книге.

Таблица 2.2. Префиксы, используемые в именах объектов

Объект Access	Префикс
таблица	tbl
запрос	qry
форма	frm
отчет	rpt
флажок	chk
комбинированный список	cbo
командная кнопка	cmd
метка	lbl
обычный список	lst
переключатель	opt
подформа/подотчет	sub
текстовое поле	txt

Зачем же нужна такая сложная схема именования? По мере написания программ вы поймете все удобство следования таким соглашениям. И, что самое важное, вы всегда с первого взгляда сможете определить, с данными или объектом какого типа вы имеете дело, а это значит довольно много при отладке или сборке программ.

Таблица 2.3. Префиксы, используемые в именах переменных

Тип данных	Префикс
байт	byt
короткий целочисленный	int
короткий с плавающей точкой	sng
длинный целочисленный	lng
длинный с плавающей точкой	dbl
текстовый	txt
валютный	cur
даты/времени	dtm
булев	boo

Применение отступов в тексте программ

Вы, наверное, уже обратили внимание, что в тексте примеров программ в настоящей главе были использованы отступы. Это — еще один хороший навык, облегчающий чтение программ ввиду наглядности их структуры.

К примеру, все строки в описанной ранее процедуре (см. рис. 2.8) имеют небольшой отступ относительно крайних строк: с именем процедуры и оператором End. Эти отступы облегчают чтение кода; к тому же программа поможет вам применять отступы автоматически.

Вы можете вставлять отступы с помощью интерфейса редактора VBE. Для этого выделите нужный фрагмент программы и выполните, соответственно, команду меню Edit⇒Indent (Применить отступ в один знак табуляции) или Edit⇒Outdent (Убрать отступ в один знак табуляции). По умолчанию знак табуляции соответствует четырем пробелам.

Совет

Отступы являются ключом к пониманию структуры программы. Еще один способ разграничения отдельных действий в программе — вставка пустых строк в логических точках. Эти своеобразные контрольные точки значительно ускорят поиск нужных логических фрагментов программы.

Комментарии

Внесение в текст программы *комментариев*, словесно описывающих выполняемые в ней действия, является хорошей практикой. Для того чтобы добавить комментарий, нужно отделить его от основного текста знаком апострофа ('), как мы уже это делали ранее в этой главе:

```
Public Sub OpenClientForm()  
    'Открытие клиентской формы  
    DoCmd.OpenForm "Clients"
```



```
Debug.Print "Форма открыта"  
End Sub
```

Каждая процедура должна иметь вступительный комментарий, описывающий ее предназначение, а также тип и значение передаваемых аргументов. Сам текст процедуры обычно складывается из множества строк и, как правило, требует большего числа комментариев. Обычно с опытом приходит умение кратко и понятно описывать производимые в программе действия, следующие рекомендации помогут вам в этом.

- Будьте кратки, насколько это возможно.
- Используйте наиболее понятные описания.
- Грамматически корректно формируйте фразы и предложения.
- Правильно используйте знаки пунктуации.
- Избегайте комментариев, повторяющих текст программы.
- Комментируйте передаваемые аргументы, если их атрибуты не очевидны в тексте программы.
- Комментируйте версии фрагментов программы — кто и когда выполнил изменения и в каких целях это было сделано. (Такие комментарии вставляют далеко не все разработчики.)
- Вносите комментарии непосредственно при вводе текста программ, пока постановка задачи и ее реализация еще свежа в памяти.

Еще один тип комментариев вносится непосредственно в строках программы справа от операторов. Этот формат мы не будем использовать в настоящей книге, однако обращайтесь на него внимание, когда вам придется разбирать текст чужих программ.

Панель инструментов редактирования редактора VBE содержит кнопку комментирования фрагментов программы (это позволяет быстро отключать и включать фрагменты кода). Для ознакомления с этим процессом вначале отобразите на экране панель инструментов редактирования (**View**⇒**Toolbars**⇒**Edit**). После этого выделите фрагмент текста программы и щелкните на кнопке **Comment Block** (Закомментировать фрагмент) данной панели инструментов. На рис. 2.16 показан результат комментирования всей процедуры, выполненной с помощью одного щелчка мыши. Для того чтобы снять комментарий, щелкните на кнопке **Uncomment Block** панели инструментов редактирования.

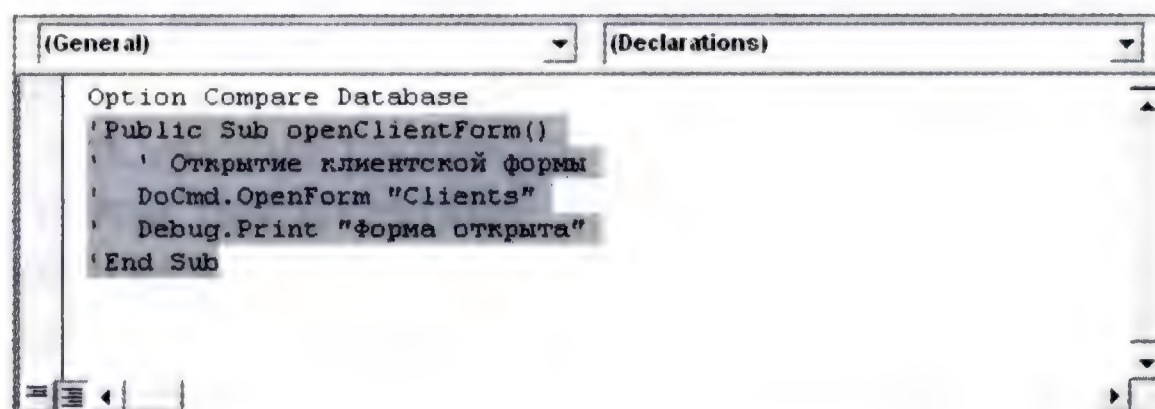


Рис. 2.16. Быстрое комментирование целой процедуры

Переменные, константы и типы данных

3

Объявление переменных и констант

Процесс изучения языков программирования и, в частности, VBA ничем не отличается от изучения иностранных языков — сперва нужно ознакомиться с основами. Язык VBA, как и любой другой, имеет собственный *синтаксис* — правила комбинирования компонентов языка, позволяющие программе Access (или другому приложению) понять инструкцию и ответить на нее. Изучив эти правила, вы без труда сможете общаться с Access.

В языке VBA для представления значений используются переменные и константы. С технической точки зрения *переменной* называется небольшая область памяти, в которой хранится некоторая информация; в то же время внешне — это всего лишь имя, за которым скрывается эта информация. *Константы* подобны переменным — они также предназначены для хранения данных. Основное различие между ними состоит в том, что константы представляют неизменяемые значения, в то время как данные в переменных могут в любое время быть изменены.

В ЭТОЙ ГЛАВЕ

Объявление переменных и констант	51
Типы данных в VBA	57
Синтаксис ссылок	62

Объявление переменных

Переменные представляют в программе значения или объекты. Используя соглашения об именах, им присваивают некоторое описательное название, задают тип хранимых данных, после чего используют для хранения и изменения данных.

Перед использованием переменной ее нужно объявить с помощью оператора `Dim`, имеющего следующий синтаксис:

```
Dim имя_переменной As [New] тип_данных
```

где *имя_переменной* задается пользователем согласно используемым соглашениям, а *тип_данных* является одним из типов, используемых в VBA. Если опустить в объявлении тип данных, по умолчанию будет использован тип `Variant`; однако рекомендуется все же четко задавать тип при объявлении. Переменные типа `Variant` требуют для своего хранения немного больше памяти; к тому же работа с ними немного замедляет выполнение программы. С другой стороны, такие переменные не ограничены хранением какого-то одного типа данных. На практике довольно редко встречается ситуация, когда в одной переменной может храниться несколько типов данных, а это и есть условие оптимальности применения типа `Variant`.

➔ Ключевое слово `New` не является обязательным и применимо только при объявлении переменных объектов. Этот вопрос более подробно рассмотрен в разделе “Понятие объекта” главы 8.

Совет

Как правило, переменные описываются в начале процедур. Такая компоновка не обязательна, однако большинство разработчиков стараются ее придерживаться. Группируя объявления в начале процедуры, вы можете быстрее их отыскать в теле программы.

В одной строке оператора `Dim` можно объявить несколько переменных, разделив их запятыми, например:

```
Dim переменная1 As тип_данных_1, переменная2 As тип_данных_2
```

Совет

Объявленные переменные поддерживаются функцией `IntelliSense`. Это значит, что впоследствии эти переменные можно будет выбирать из списка, а не вводить вручную, в результате чего можно будет избежать опечаток.

Использование оператора `Option Explicit`

По умолчанию язык VBA позволяет использовать в тексте программ необъявленные переменные. Чтобы проиллюстрировать этот тезис, откроем стандартный пустой модуль и введем в него следующие строки:

```
Private Function DeclarationTest()  
    varValue="Необъявленная переменная"
```



```
Debug.Print varValu  
End Function
```

Теперь при необходимости откройте окно Immediate, нажав клавиши <Ctrl+G>. После этого поместите курсор в тело процедуры и нажмите клавишу <F5>, запуская ее на выполнение. Вы, естественно, ожидаете теперь увидеть строку Неопределенная переменная в окне Immediate и удивляетесь, почему ее там нет? Все дело в том, что в операторе Debug.Print была допущена опечатка в имени переменной. Переменная varValue так и осталась соответствовать строке “Неопределенная переменная”, в то время как переменная varValu имеет пустое значение (Empty). Такую ошибку отыскать в программе довольно сложно, особенно если она имеет сложную структуру и большую длину.

Во избежание возникновения таких проблем в языке VBA можно установить обязательность объявления переменных перед их использованием. Для этого в области General Declarations окна модуля следует ввести оператор Option Explicit. Если это сделать в нашем примере, то после запуска процедуры на выполнение редактор отобразит окно ошибки, показанное на рис. 3.1.

Совет

Оператор Option Explicit позволяет быстро находить опечатки в именах переменных и объектов. Все, что требуется от вас — это включить его. Настоятельно рекомендуется включать этот режим во всех программах, так как это позволит сэкономить массу времени во время поиска опечаток, которые нередко при набивке больших программ даже профессионалами.

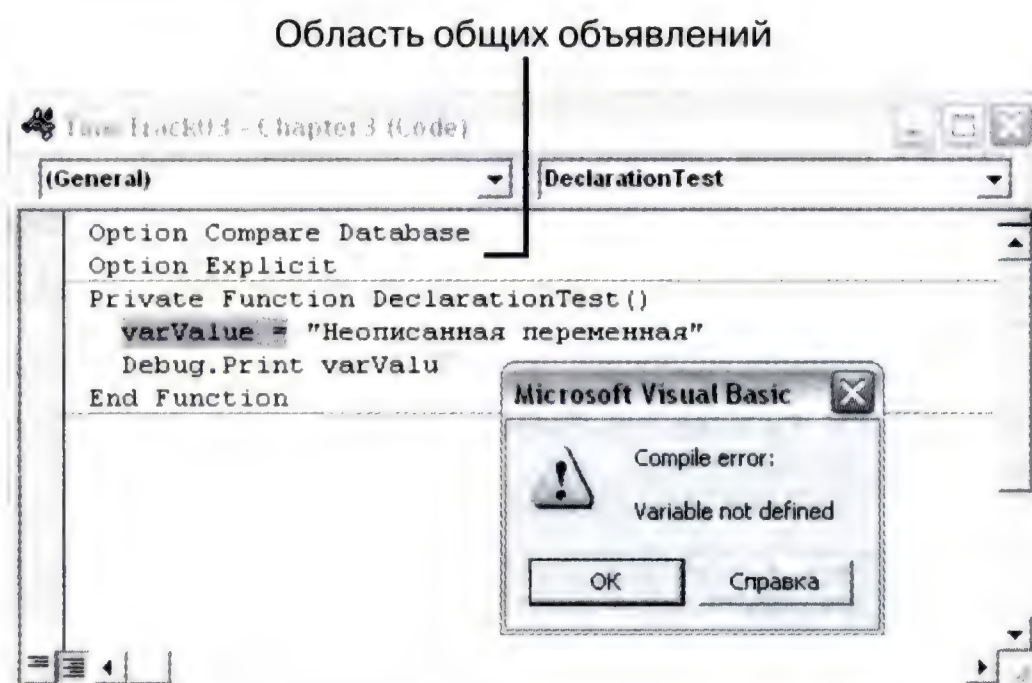


Рис. 3.1. При включенном режиме Option Explicit использование необъявленных переменных приводит к ошибке

Чтобы закрыть сообщение об ошибке, щелкните на кнопке ОК, после чего щелкните на кнопке останова программы и исправьте ошибку. Объявите переменную varValue в начале процедуры следующей строкой:

```
Dim varValue as Variant
```


после чего снова запустите программу на выполнение. На этот раз ошибка возникнет, когда вы дойдете до имени переменной, в котором допущена опечатка (рис. 3.2). Снова щелкните ОК на кнопке останова программы и исправьте опечатку в переменной `varValue`. Если теперь вы запустите программу на выполнение, то в окне **Immediate** увидите долгожданную строку (рис. 3.3).

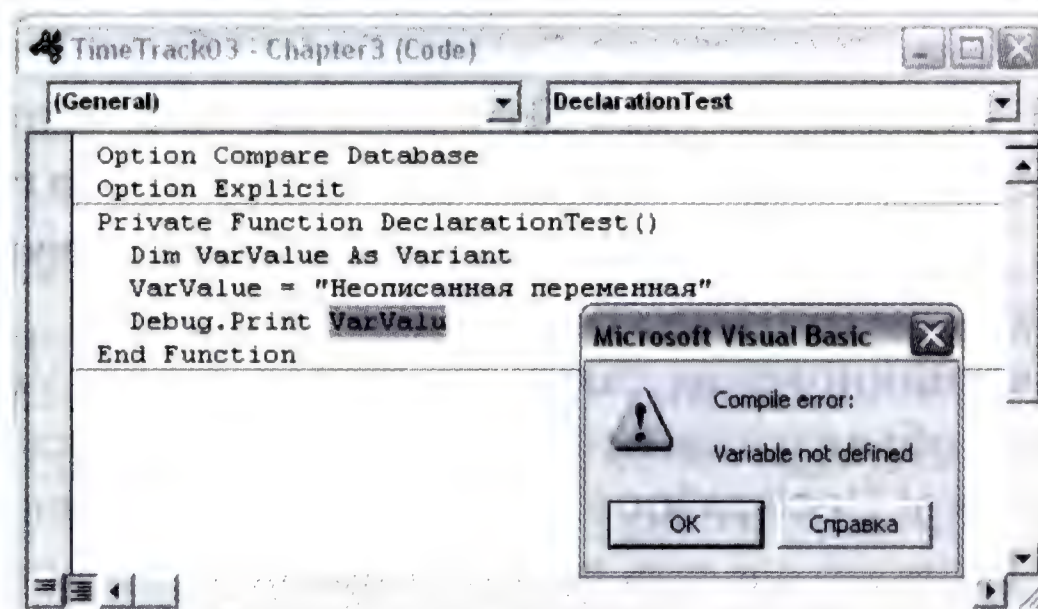


Рис. 3.2. Редактор VBA нашел переменную с опечаткой

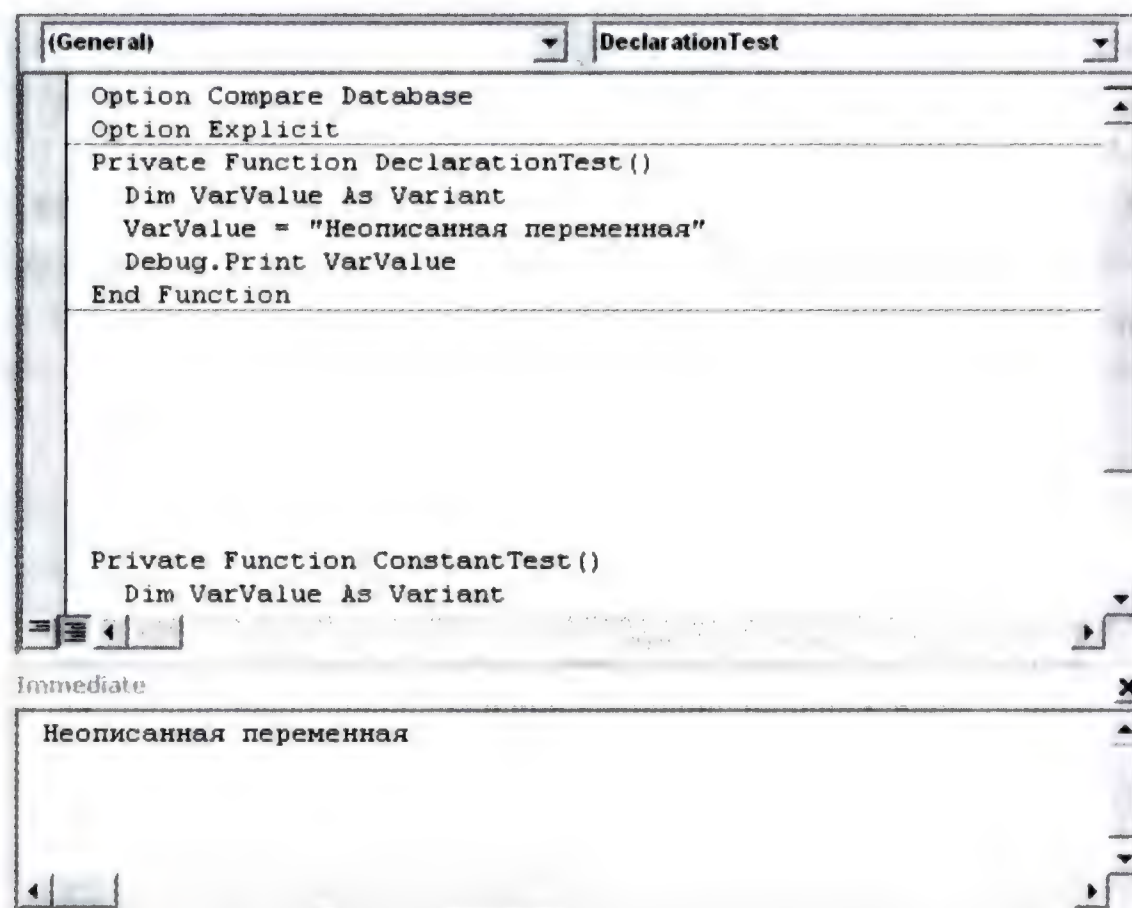


Рис. 3.3. После исправления опечатки процедура выполняется корректно

Описанный выше метод устанавливал обязательное объявление переменных только для одного модуля. Чтобы активизировать этот режим для всех новых модулей, нужно выполнить следующие действия:

1. В редакторе VBE выберите в меню пункт **Tools**⇒**Options**.
2. Перейдите к закладке **Editor**.
3. Установите флажок **Require Variable Declaration** (рис. 3.4).
4. Щелкните на кнопке **OK** и закройте диалоговое окно параметров.

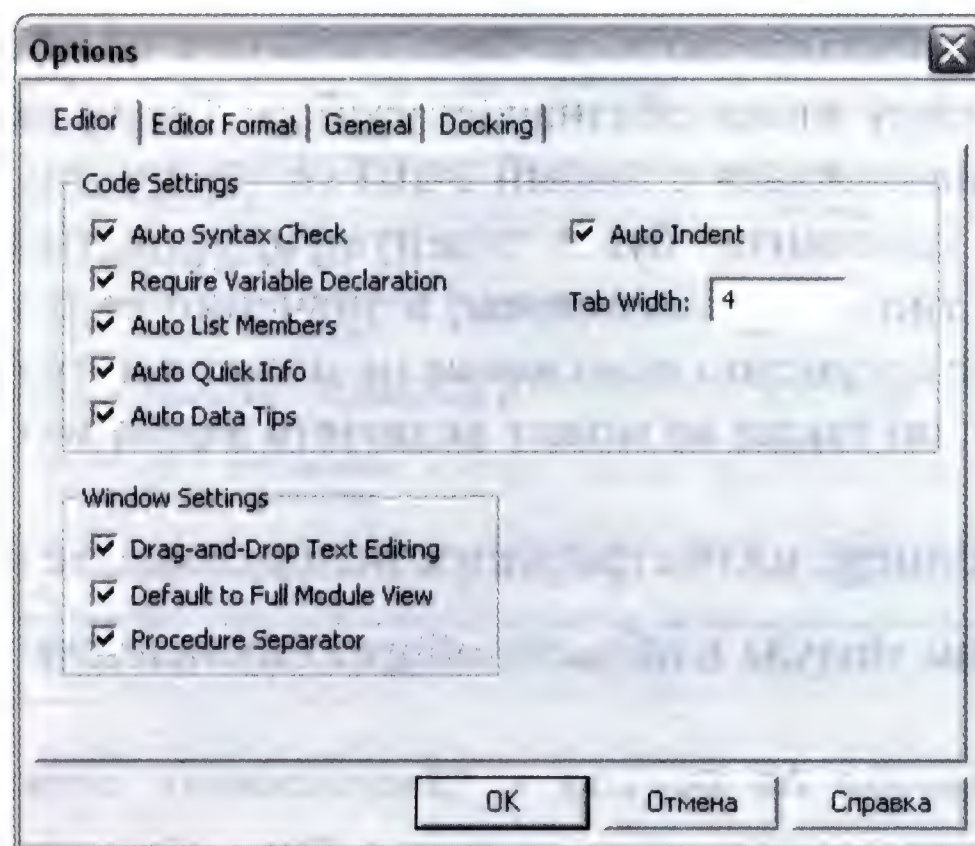


Рис. 3.4. Включите режим автоматической проверки объявлений переменных

Предупреждение

Включение режима проверки объявлений в окне параметров редактора влияет только на модули, созданные после его активизации. Во все уже существующие модули придется добавить оператор `Option Explicit` вручную.

Имена переменных

Соглашение об именах касается также и переменных. Однако кроме установленных вами правил существуют и другие, выполнение которых обязательно.

- Имя переменной должно начинаться с буквы.
- В имени переменной нельзя использовать специальные символы, такие как . (точка), , (запятая), %, \$, !, # и @.
- Имена переменных должны быть уникальными. Другими словами, две разные процедуры или переменные не могут иметь одно и то же имя.
- Длина имени переменной не должна превышать 255 символов.

➔ Соглашения об именах рассматриваются в разделе “Выработка навыков по вводу кода” главы 2.

Объявление констант

Следует обратить внимание, что в языке VBA термин *константа* имеет несколько значений. И константы и переменные представляют некоторое значение объекта. Разница состоит в том, что значение константы в ходе выполнения программы не может измениться, даже по ошибке.

Для объявления константы используется оператор `Const`, имеющий следующий синтаксис:

```
[Public | Private] Const имя_константы As тип_данных = выражение
```


Ключевые слова в квадратных скобках являются не обязательными, а вертикальная черта между ними обозначает выбор. Таким образом, объявление константы может начинаться словами `Public`, `Private` или просто `Const`. В любом случае, константа будет идентифицироваться по имени *имя_константы*, иметь тип *тип_константы*, а значение ее будет результатом *выражения*. В качестве аргумента *выражение* не может выступать переменная или результат функции; он также не может включать в себя множество встроенных функций VBA.

В следующем примере иллюстрируется использование констант.

1. В стандартном модуле в области общих объявлений введите следующий оператор:

```
Const conMessage As String = "Неописанная переменная"
```

2. С помощью диалогового окна `Insert Procedure` или вручную введите следующую процедуру:

```
Private Function ConstantTest()  
    dim varValue as Variant  
    varValue = conMessage  
    Debug.Print varValue  
End Function
```

3. Установив курсор в любое место процедуры `ConstantTest()`, нажмите клавишу `<F5>`, запустив функцию на выполнение. Как видно на рис. 3.5, оператор `Debug.Print` напечатал значение переменной `varValue`, равное константе `conMessage`.

Теперь измените значение константы на строку “Константа” и снова запустите процедуру на выполнение. На этот раз значением переменной стала уже строка “Константа” (рис. 3.6). Теперь представьте себе, что в тексте длинной программы вам нужно множество раз использовать эту строку. Вместо того, чтобы при каждом изменении фразы изменять каждое из ее вхождений, теперь достаточно изменить значение константы `conMessage` в объявлении `Const`.

Встроенные константы

В дополнение к определению произвольных констант, язык VBA предлагает целый ряд предопределенных констант, известных как *встроенные константы*. Они используются для определения специфических типов данных, а также в других целях.

В использовании встроенных констант скрыто два существенных достоинства. Во-первых, не нужно объявлять их вручную. Во-вторых, они повышают понятность текста программы. Имена встроенных констант описывают их значения или назначения, что делает их легко узнаваемыми. Для назначения определенных значений переменной можно использовать соответствующие встроенные константы, например:

```
varValue = vbEmpty
```

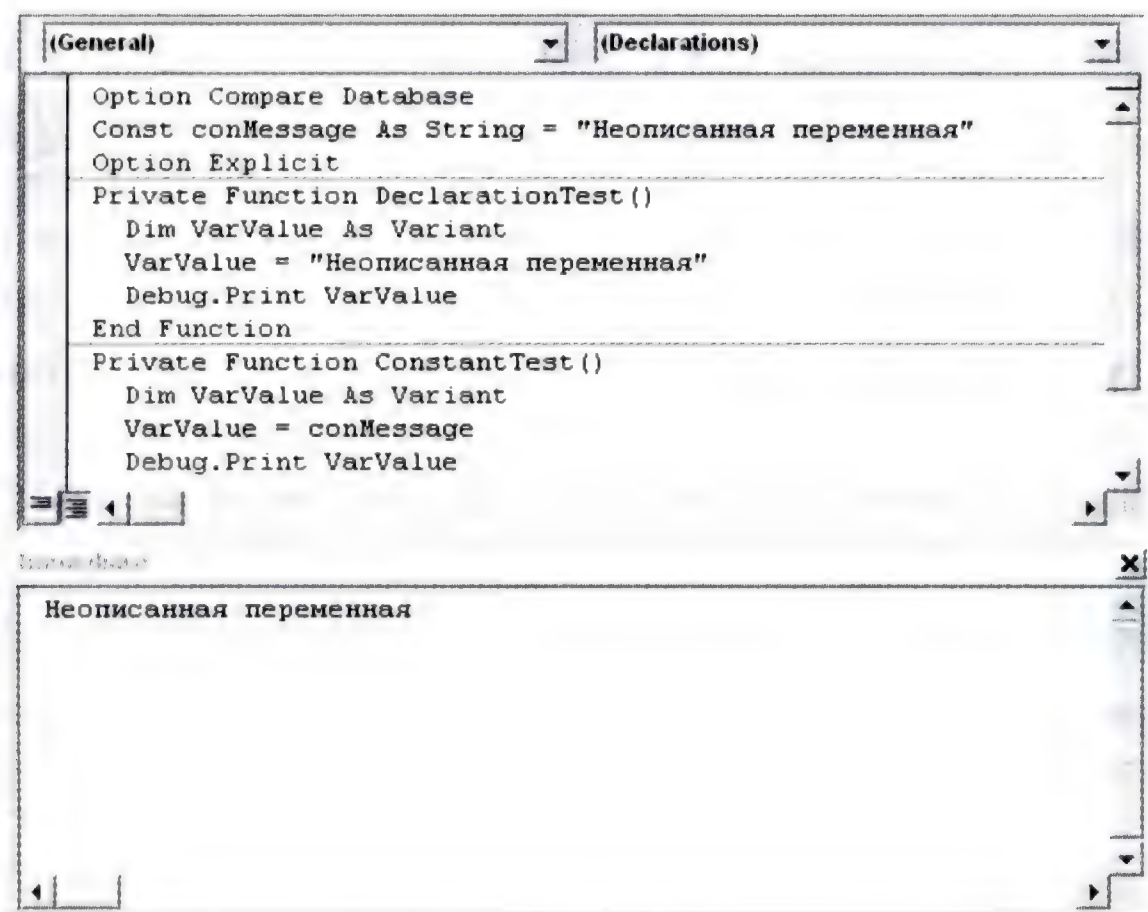



Рис. 3.5. Переменная VarValue равна константе с именем conMessage

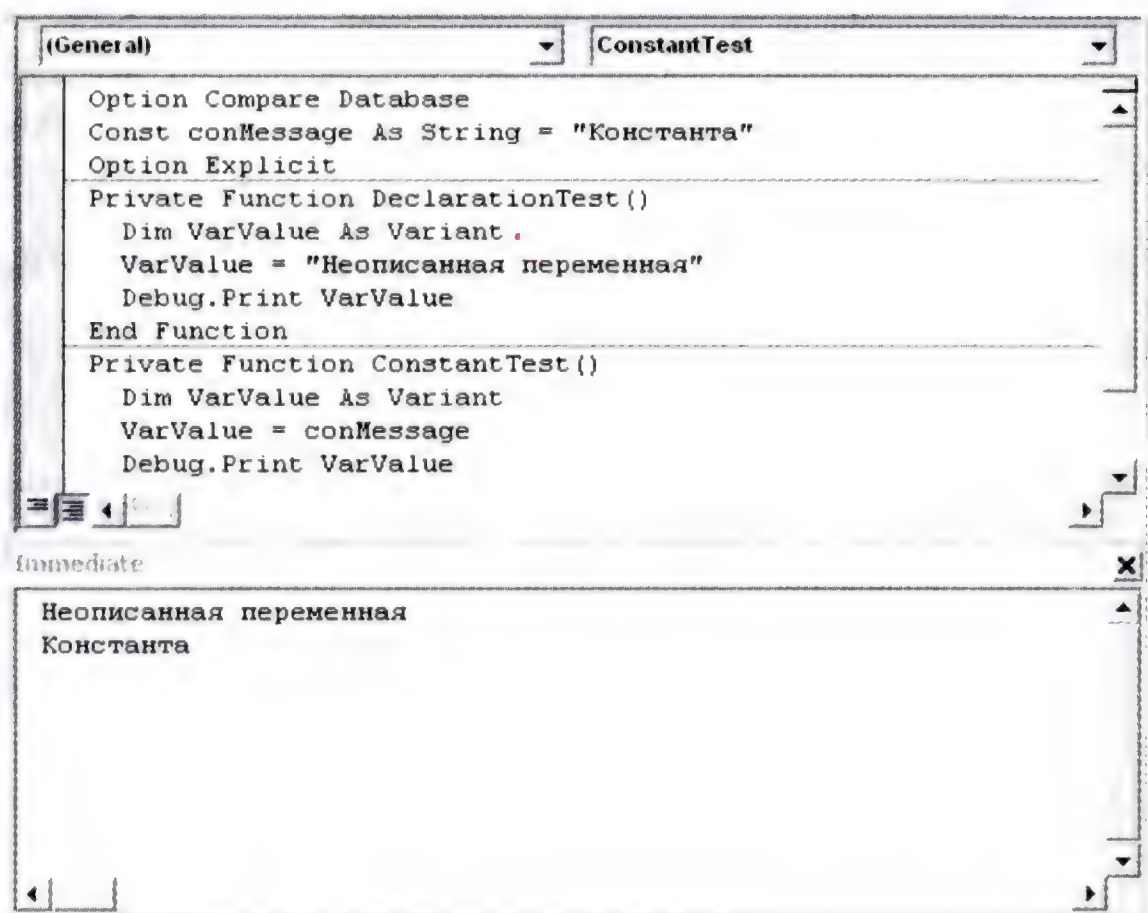


Рис. 3.6. Результат изменения константы conMessage

Встроенные константы предопределены в языке VBA; вам не нужно объявлять их в тексте программы, чтобы вставить их имена в выражения и операторы. Встроенные константы более подробно рассматриваются в следующем разделе.

Типы данных в VBA

При объявлении переменной можно также определять и ее тип данных. С некоторыми типами данных вы уже знакомы, так как назначали их табличным полям в базах данных Access. Язык VBA для определения типов переменных использует те же типы данных.

Наиболее важной функцией типа данных является проверка достоверности самих данных. Задание определенного типа данных не избавит вас от ввода неправильного значения, однако полностью предотвратит ввод некорректных типов данных. Если при объявлении переменной опустить ее тип, к ней будет применен тип `Variant`. Этот тип данных является самым гибким, и он отдает на откуп самому интерпретатору языка VBA определение наиболее подходящего типа данных для переменной. В табл. 3.1 приводится сравнительная характеристика различных типов данных, используемых в языке VBA.

Таблица 3.1. Сравнительная характеристика типов данных VBA				
Тип данных	Требуемая память	Значение по умолчанию	Константа VBA	Диапазон данных
Целое	2 байта	0	<code>vbInteger</code>	От -32 768 до 32 767
Длинное целое	4 байта	0	<code>vbLong</code>	От -2 147 483 648 до 2 147 486 647
Короткое с плавающей точкой	4 байта	0	<code>vbSingle</code>	От -3402823E38 до -1,401298E-45 или от 1,401298E-45 до 3402823E38
Длинное с плавающей точкой	8 байт	0	<code>vbDouble</code>	От -1,79769313486232E308 до -4,94065645841247E-324 или от 4,94065645841247E-324 до 1,79769313486232E308
Валюта	8 байт	0	<code>vbCurrency</code>	От -922337203685477,5808 до 922337203685477,5807
Дата	8 байт	00:00:00	<code>vbDate</code>	От 1 января 100 года до 31 декабря 9999 года
Фиксированная строка	Длина строки	Строка из пробелов соответствующей длины	<code>vbString</code>	От 1 до 65400 символов
Переменная строка	10 байт плюс количество символов	Пустая строка ("")	<code>vbString</code>	От 0 до 2 миллиардов символов

Окончание табл. 3.1

Тип дан-ных	Требуемая память	Значение по умолчанию	Константа VBA	Диапазон данных
Объект	4 байта	Ничего (vbNothing)	vbObject	Любой объект Access, или компонент ActiveX, или объект Class
Булев тип	2 байта	False	vbBoolean	–1 или 0
Variant	16 байт	Пусто (vbEmpty)	vbVariant	То же, что и длинный с плавающей точкой
Десятич-ный	14 байт	0	vbDecimal	От –79228162514264337593543950335 до 79228162514264337593543950335 или от –7,9228162514264337593543950335 до 7,9228162514264337593543950335
Байт	1 байт	0	vbByte	От 0 до 255

Булев тип данных Boolean

Этот тип данных используется для хранения логических данных, которые могут принимать только одно из двух значений: включено или выключено, правда или ложь, да или нет и т.п. Ключевые слова True и False являются предопределенными константами, имеющими, соответственно, значения –1 и 0. Для иллюстрации этих ключевых слов введите по одному нижеследующие выражения в окно Immediate (рис. 3.7):

```
?True = 0
?True = -1
?False = 0
?False = -1
?True = False
```



Рис. 3.7. Константы True и False равны, соответственно, –1 и 0

Тип данных Byte

Это — самый компактный тип данных VBA, позволяющих хранить целочисленные неотрицательные значения от 0 до 255. Если попытаться присвоить переменной этого типа недопустимое значение, система выдаст сообщение об ошибке.

Валютный тип данных Currency

Валютный тип данных используется для хранения монетарных значений от -922 337 203 685 477,5808 до 922 337 203 685 477,5807. Результатом операций с данными типа Currency является значение с точностью 14 знаков до запятой и 4 знаками после. Этот тип данных используется для предотвращения ошибок округления, когда точность имеет первостепенную важность.

Тип данных Date

Этот тип данных позволяет хранить специальным образом отформатированные числовые значения, которые представляют как дату, так и время. Нет никакой необходимости отдельно хранить и дату, и время — этот тип данных позволяет хранить как одно из этих значений, так и оба сразу. Возможно хранение данных в диапазоне от 1 января 100 года до 31 декабря 9999 года.

Десятичный тип данных Decimal

Десятичный тип данных является подтипом типа Variant и не разделяет типы данных, принимая значения от -79 228 162 514 264 337 593 543 950 335 до 79 228 162 514 264 337 593 543 950 335, если они не содержат десятичных знаков. В то же время этот тип позволяет хранить данные с точностью до 28 десятичных знаков для значений от -7,9228162514264337593543950335 до 7,9228162514264337593543950335.

Числа с плавающей точкой двойной точности (Double)

Этот тип данных используется для хранения дробных чисел, когда точность играет первоочередную роль. Диапазон допустимых значений составляет от -1,79769313486232E308 до -4,94065645841247E-324 и от 4,94065645841247E-324 до 1,79769313486232E308.

Целочисленный тип данных Integer

Этот тип данных в программах используется чаще всего наряду со строковым типом. Он позволяет хранить целые числа в диапазоне от -32 768 до 32 767.

Длинный целочисленный тип данных Long

Как и тип Integer, этот тип позволяет хранить целочисленные данные, однако в значительно более широком диапазоне: от -2 147 483 648 до 2 147 486 647.

Объектный тип данных Object

На самом деле переменная типа `Object` является всего лишь ссылкой на объект `Access`, такой как форма, отчет или элемент управления. Этот тип переменной может быть и ссылкой на объект `ActiveX` либо на объект класса, созданный в модуле класса.

- ➔ Модули класса кратко рассмотрены в разделе “Введение в модули VBA” главы 2, более подробно они описаны в главе 8.
- ➔ Переменные типа `Object` подробно рассмотрены в главе 8.

Тип данных Single

Этот тип данных позволяет хранить дробные числа. Этот тип аналогичен типу `Double`, однако имеет более узкий диапазон значений: от $-3402823E38$ до $-1,401298E-45$ и от $1,401298E-45$ до $3402823E38$.

Строковый тип данных String

Строковый тип данных — один из наиболее часто используемых в VBA. В нем можно хранить как буквы, так и числа, рассматривая их как текст. Существует два типа строковых значений: фиксированной и переменной длины; первая может содержать от 1 до 65400 символов, вторая — от нуля и до 2 миллиардов. Для объявления строки фиксированной длины используется оператор `Dim` следующей конструкции:

```
Dim имя_переменной As String * длина_строки
```

В противоположность этому, строки переменной длины могут расширяться настолько, насколько этого требуют данные. По умолчанию все строковые данные относятся к этому подтипу. Для объявления этого типа используется следующий оператор:

```
Dim имя_переменной As String
```

Тип данных Variant

Переменные этого типа могут хранить как числовые, так и не числовые данные. Этот тип данных является наиболее гибким, поскольку позволяет хранить очень большие значения практически любого типа (по длине он соответствует типу `Double`). Его используют только в том случае, когда точно предсказать заранее тип данных трудно, или когда эти данные принимаются из внешних источников, спецификации которых у вас нет.

Тип данных `Variant` принят в языке VBA по умолчанию, поэтому в следующем фрагменте переменная `varValue` будет интерпретироваться системой как имеющая тип `Variant`:

```
Dim VarValue
```

Несмотря на свою гибкость, использование типа `Variant` приводит к замедлению процесса обработки данных, так как частью его работы является

определение наиболее подходящего типа для данного значения и выполнение соответствующего преобразования. В то же время, это замедление вряд ли будет заметно на современных быстродействующих компьютерах.

Самым большим минусом этого типа данных является его логическая неопределенность в тексте программы. Невозможно заранее определить, с каким типом данных придется иметь дело, к каким последствиям это может привести и не станет ли это источником ошибки.

Синтаксис ссылок

Значительное место в ваших программах будут занимать ссылки на объекты Access — формы и элементы управления. Знание того, как корректно ссылаться на объекты и данные, является одним из основных признаков мастерства. Для начала ознакомимся с некоторыми терминами.

- *Идентификатор*. Это набор символов, который идентифицирует значение элемента управления, свойства или другого выражения.
 - *Оператор*. В данном контексте это — символ, используемый для разделения отдельных компонентов идентификатора. Идентификатор может иметь несколько уровней. Существует два оператора идентификатора: точка и восклицательный знак. В этом разделе вы детально ознакомитесь с ними.
 - *Спецификатор*. Он идентифицирует набор объектов.
- ➔ Более подробно об операторе “точка” читайте в разделе “Чтение и установка свойств” главы 8.

При ссылке на объекты интерпретатору VBA необходимо знать не только местонахождение объекта, но и то, какого типа этот объект. В базе данных `TimeTrack.mdb`, используемой в настоящей книге в качестве примера, существует несколько объектов — таблиц, форм, отчетов и модулей VBA. Формы и отчеты имеют в своем составе более 60 элементов управления. Таким образом, число возможных ссылок в базе данных значительно.

Для ссылки на любой из этих объектов используется следующая форма:

спецификатор! [*имя_объекта*]

где *спецификатор* идентифицирует набор объектов, а *имя_объекта* — сам объект. Обратите внимание на символ восклицательного знака, разделяющий эти два компонента. К примеру, для ссылки на форму `Clients` базы данных `TimeTrack.mdb` используется следующее выражение:

`Forms! [Clients]`

Здесь составляющая `Forms` указывает на набор форм, а `Clients` — на имя самой формы. А следующее выражение ссылается на отчет счета-фактуры:

`Reports! [BillingReport]`

Синтаксис ссылок на элементы управления немного отличается, так как любой из них принадлежит некоторой форме или отчету (*родительскому объ-*

екту). Это значит, что для доступа к ним нужно пройти два других уровня и использовать следующую форму:

спецификатор! [имя_объекта]! [имя_элемента]

где *спецификатор* принимает значение Forms (формы) или Reports (отчеты).

Предупреждение

Элементы управления отображают данные, извлеченные из полей таблиц или запросов. Термином *поле* называют столбец в таблице или запросе, в котором хранятся данные. *Элементом управления* называют объект, отображающий эти данные в форме или отчете. Часто элементы управления имеют то же имя, что и соответствующие им поля, хранящие данные, и это может вас немного запутать. Большинство программистов стараются подчеркивать отличия между этими объектами, дополняя имена элементов специальными префиксами. В результате имя txtFirstName может стать именем элемента текстового поля, отображающего данные, хранимые в поле FirstName некоторой таблицы.

В качестве примера рассмотрим форму Employees, показанную на рис. 3.8. На этой форме размещены три элемента: EmployeeID (табельный номер), FirstName (имя) и LastName (фамилия). Полное имя каждого из элементов состоит из спецификатора, идентификатора формы и идентификатора самого элемента управления:

```
Forms! [Employees]! [EmployeeID]  
Forms! [Employees]! [FirstName]  
Forms! [Employees]! [LastName]
```

Все три элемента управления принадлежат коллекции Controls данной формы, принятой в форме по умолчанию. Исходя из этого, в полном имени элемента можно опустить имя коллекции элементов.

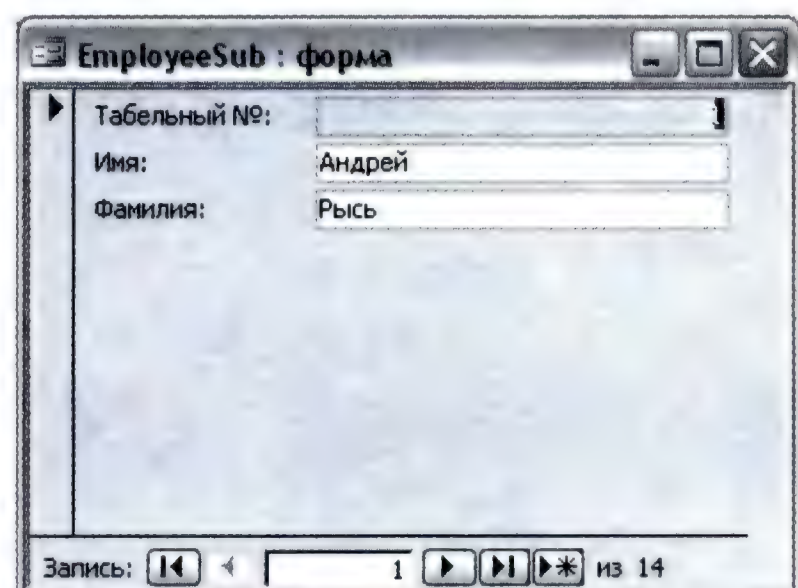


Рис. 3.8. Набор элементов управления принадлежит форме или отчету

Совет

Вы, наверное, уже заметили, что имена всех объектов заключены в квадратные скобки. Они необходимы для того, чтобы интерпретатор VBA не возвращал сообщение об ошибке, если в имени встретится символ пробела. Если в имени не содержится пробелов, квадратные скобки вполне можно опустить. В то же время, если вы забудете обрамить квадратными скобками имя, содержащее пробелы, интерпретатор языка VBA вернет сообщение об ошибке. Именно поэтому большинство программистов берут за привычку обрамлять квадратными скобками *все* идентификаторы — на всякий случай. Самым же лучшим решением будет отсутствие символов пробелов в именах объектов.

Использование процедур

4

Типы процедур

Программы на языке VBA организованы в виде набора *инструкций*. Инструкция представляет собой одну строку текста, например:

```
intCount = 6
```

В данной инструкции переменной `intCount` присваивается значение 6. Несмотря на то, что отдельные инструкции можно выполнять в окне Immediate, в модуле VBA сохранять отдельные инструкции нельзя — в них предложения должны быть организованы в *процедуры*. Процедурой называют группу инструкций, совместно выполняющих некоторую задачу. В предыдущих главах были представлены примеры небольших процедур; в этой же главе мы более глубоко изучим вопросы создания и использования процедур.

Для начала рассмотрим, в чем же различие между двумя типами процедур:

- *подпрограммами* (часто называемыми собственно *процедурами*) и
- *функциями*.

В ЭТОЙ ГЛАВЕ

Типы процедур	65
Объявление общих и частных процедур	69
Передача аргументов.....	69
Присвоение функции типа данных.....	72
Обработка ошибок	73
Отладка программ.....	75

→ Существует также и два специализированных типа процедур — *процедуры свойств* и *процедуры обработки событий*. Информацию об этих типах вы найдете в соответствующих разделах главы 8.

Создание и использование подпрограмм

Подпрограммой называют процедуру, которая не возвращает значение (о возвращении значений читайте в следующем разделе). Вот пример простой подпрограммы:

```
Sub Procedure1 ()
    'Вывод сообщения в окне Immediate
    Debug.Print "Привет!"
End Sub
```

Подпрограмма начинается с ключевого слова `Sub`, за которым следует имя процедуры (в данном примере `Procedure1`) и пустая пара круглых скобок.

Подпрограмма завершается ключевым словом `End Sub`. Между этими двумя строками вы можете ввести любое количество инструкций. В нашем примере мы вставили в процедуру комментарий и оператор `Debug.Print` (этот оператор выводит в окно `Immediate` все, что следует за ним в предложении).

Запустить подпрограмму можно, набрав ее имя в окне `Immediate` и нажав клавишу `<Enter>` (рис. 4.1).



Рис. 4.1. Запуск подпрограммы в окне Immediate

Процедуры могут содержать несколько инструкций. В этом случае интерпретатор VBA выполняет их все поочередно, сверху вниз. Для примера рассмотрим следующую подпрограмму:

```
Sub Procedure2()
    ' Сложение двух чисел
    Dim intFirst As Integer
    Dim intSecond As Integer
    intFirst = 4
    intSecond = 7
    Debug.Print intFirst + intSecond
End Sub
```

Если запустить эту процедуру в окне `Immediate`, она выведет число 11 — результат сложения чисел 4 и 7.

Как правило, процедуры редко запускают из окна `Immediate`. Чаще всего используются два других способа:

- вызов из другой процедуры;
- запуск из интерфейса пользователя программы Access.

Для запуска подпрограммы из другой процедуры нужно в последнюю вставить предложение, состоящее из имени вызываемой подпрограммы, например:

```
Sub Procedure3()  
    ' Запуск другой процедуры  
    Debug.Print "Мы в процедуре 3"  
    Procedure4  
    Debug.Print "Мы снова в процедуре 3"  
End Sub
```

```
Sub Procedure4()  
    ' Печать сообщения  
    Debug.Print "Мы в процедуре 4"  
End Sub
```

На рис. 4.2 показан результат вызова процедуры Procedure3 из окна Immediate. На печать выводится первое сообщение из главной процедуры, после чего мы переходим в процедуру Procedure4 и снова возвращаемся в главную.

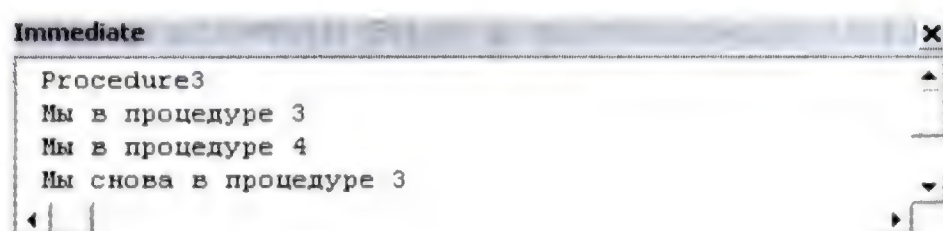


Рис. 4.2. Вызов подпрограммы из другой подпрограммы

➔ Вызов процедур из интерфейса пользователя Access будет рассмотрен в главе 10.

Создание и использование функций

Вторым типом процедур являются *функции*, которые практически во всем идентичны подпрограммам за исключением одного — они возвращают значение. Для иллюстрации рассмотрим следующий пример:

```
Function Procedure5()  
    Procedure5 = "Привет!"  
End Function
```

Обратите внимание, что функция начинается с ключевого слова `Function`, за которым следует имя процедуры и пустая пара круглых скобок; заканчивается же функция инструкцией `End Function`. Между этими двумя строками, как и в подпрограмме, может находиться любое количество инструкций. Однако в функции существует специальный тип предложения, которое свойственно только ей. Это — инструкция *возврата значения*. В данном случае ею является единственное предложение функции Procedure5, в котором мы присваиваем имени процедуры строку “Привет!”. Любая функция возвращает значение. Если это значение не указывается явным образом, возвращается специальное значение `Null`, являющееся одним из типов пустого значения.

На рис. 4.3 показан результат запуска функции Procedure5 в окне Immediate.

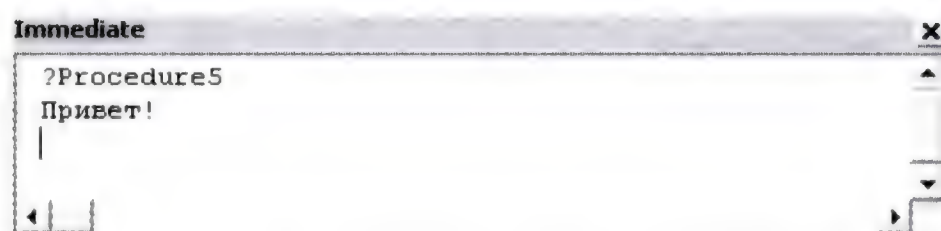


Рис. 4.3. Вызов функции в окне Immediate

Обратите внимание на знак вопроса, стоящий перед названием функции в окне Immediate. Он указывает интерпретатору VBA на то, что требуется выполнить функцию и вывести на экран возвращаемое ею значение. В результате интерпретатор выводит на экран строку с возвращаемым значением даже несмотря на отсутствие оператора `Debug.Print`.

Совет

Для выполнения простых вычислений в окне Immediate можно также использовать оператор вопросительного знака. К примеру, если набрать в этом окне `?12*12` и нажать <Enter>, будет напечатан результат этой операции – 144.

Подобно подпрограммам, функции могут содержать больше одной инструкции и могут вызываться из других процедур, например:

```
Function Procedure6()  
    ' Вызов функции из другой функции  
    Debug.Print "Мы в процедуре 6"  
    Procedure7  
    Debug.Print "Мы снова в процедуре 6"  
End Function
```

```
Function Procedure7()  
    ' Печать сообщения  
    Debug.Print "Мы в процедуре 7"  
End Function
```

С помощью функций можно осуществить возврат в вызывающую процедуру некоторого значения, например:

```
Function Procedure8()  
    ' Использование значения, полученного в другой процедуре  
    Dim i As Integer  
    i = Procedure9  
    Debug.Print i  
End Function  
  
Function Procedure9()  
    Procedure9 = 5  
End Function
```

Если запустить функцию Procedure8 в окне Immediate, на печать будет выведено число 5. Это происходит потому, что результат выполнения функции Procedure9 присваивается в функции Procedure8 переменной `i`, значение которой, в свою очередь, выводится на печать.

Объявление общих и частных процедур

Редко можно встретить процедуры, объявленные только с использованием ключевых слов `Sub` или `Function` — перед ними, как правило, находятся различные *модификаторы*. Самыми важными модификаторами являются `Public` (общий) и `Private` (частный). Общие процедуры могут быть вызваны из любого места программы на VBA; частные же — только из модуля, в котором были объявлены. Вот как выглядят объявления этих процедур (здесь сами процедуры ничего не выполняют).

```
Public Sub Procedure10()  
    ' Может быть вызвана из любого места  
End Sub  
  
Private Sub Procedure11()  
    ' Может быть вызвана только из этого модуля  
End Sub  
  
Public Function Procedure12()  
    ' Может быть вызвана из любого места  
End Function  
  
Private Function Procedure13()  
    ' Может быть вызвана только из этого модуля  
End Function
```

Предупреждение

Частные процедуры нельзя вызывать в окне Immediate.

Передача аргументов

До сих пор все процедуры, с которыми мы работали, выполняли при каждом вызове в точности одно и то же. Однако такое происходит крайне редко. Обычно в процедуры передают аргументы, влияющие на ход их выполнения. При определении процедуры (с использованием ключевых слов `Sub` или `Function`) можно указать также и список принимаемых ею *аргументов*. Вот пример процедуры, принимающей только один аргумент.

```
Sub Procedure14(intInput As Integer)  
    ' Удваивает вводимый аргумент и выводит на печать результат  
    Debug.Print (intInput * 2)  
End Sub
```

В объявлении процедуры `Procedure14` указано, что она принимает один аргумент типа `Integer`. Это значение присваивается переменной `intInput`, определенной внутри процедуры. Далее на печать выводится переданное значение, умноженное на два. На рис. 4.4 показан результат вызова этой процедуры с передачей разных значений аргумента.

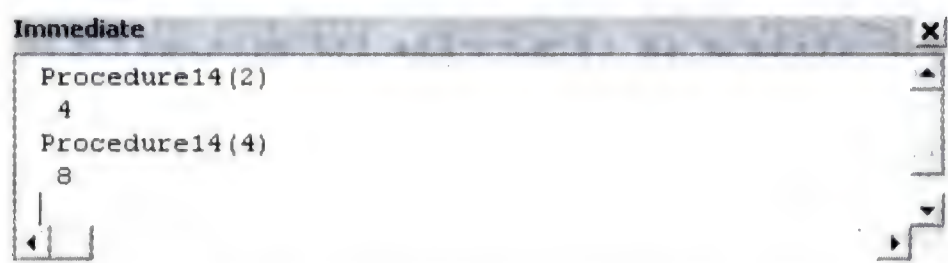


Рис. 4.4. Вызов процедуры с одним аргументом

➔ Список типов данных, которые можно использовать при передаче аргументов, содержится в разделе “Типы данных в VBA” главы 3.

Процедуры могут иметь больше одного аргумента. Вот пример функции, которой передаются два строковых аргумента и которая возвращает строку, являющуюся их объединением.

```
Function Procedure15(strIn1 As String, strIn2 As String)
    ' Объединение пары строк
    Procedure15 = strIn1 & strIn2
End Function
```

Примечание
Оператор *слияния* (или *конкатенации*) объединяет две строки в одну.

Когда в процедуре описывается больше одного аргумента, их разделяют запятыми.

Использование необязательных аргументов и значений по умолчанию

В объявлении процедуры можно указать, что некоторые аргументы не являются обязательными. Для примера рассмотрим процедуру, имеющую один необязательный аргумент.

```
Sub Procedure16(Optional strIn As String)
    ' Печать аргумента, если такой существует
    Debug.Print (strIn)
End Sub
```

На рис. 4.5 показано, что такую процедуру можно вызывать как с аргументом, так и без него; в последнем случае внутренней переменной аргумента ничего не присваивается.

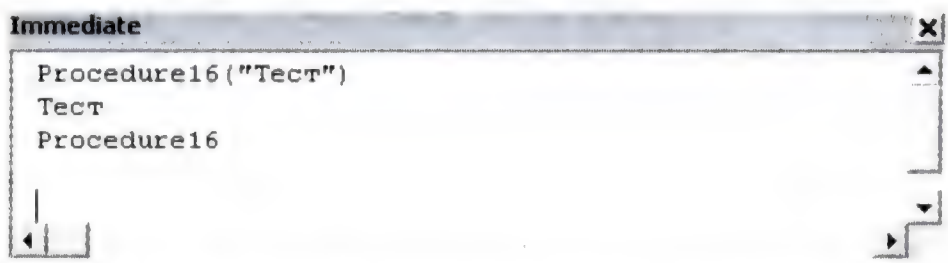


Рис. 4.5. Вызов процедуры с необязательным аргументом

Необязательный аргумент может иметь значение, присваиваемое ему по умолчанию, если при вызове процедуры этому аргументу не передается конкретное значение. Рассмотрим пример такой процедуры.


```
Sub Procedure17(Optional strIn As String = "Отсутствует")
    ' Печать аргумента или значения по умолчанию
    Debug.Print strIn
End Sub
```

Если этой процедуре не передать аргумент `strIn`, ему будет присвоена строка по умолчанию — "Отсутствует".

Примечание

Передаваемые значения трактуются как и любые другие. Это значит, что вы их должны подобающим образом обрамлять. В частности, в приведенном примере переменная `strIn` имеет строковый тип `String`, поэтому при передаче в нее аргумента последний нужно заключать в кавычки.

Передача аргументов по ссылке

По умолчанию в языке VBA все аргументы передаются в процедуры *по ссылке*. Это значит, что когда переменная передается из одной процедуры в другую, то вызванная процедура работает в точности с той же копией переменной, что и вызывающая. В следующем примере показана работа с аргументом, передаваемым по ссылке.

```
Sub Procedure18()
    ' Демонстрация передачи по ссылке
    Dim i As Integer
    i = 5
    ' Передача по ссылке в другую процедуру
    Procedure19 i
    ' Печать результата
    Debug.Print i
End Sub
```

```
Sub Procedure19(intInput As Integer)
    intInput = 12
End Sub
```

Если запустить процедуру `Procedure18` в окне `Immediate`, вы увидите результат ее выполнения — число 12. Это происходит потому, что фактическая переменная с именем `i` передается в процедуру `Procedure19` по ссылке, а в последней ее значение изменяется, и после возвращения в процедуру `Procedure18` значение этой переменной остается измененным.

Совет

При передаче аргументов в подпрограмму заключать их в круглые скобки не обязательно.

Передача аргументов по ссылке работает быстрее, но может привести к побочным эффектам, если вы не ожидаете, что вызываемая процедура их будет изменять. В этом случае вы можете воспользоваться еще одним способом передачи аргументов — *по значению*.

Передача аргументов по значению

При передаче аргументов из одной процедуры в другую по значению интерпретатор VBA создает копию переменной, с которой в дальнейшем работает вызываемая процедура. Таким образом, все изменения, проводимые с переданной переменной в вызванной процедуре, никак не отражаются на значении этой переменной в вызывающей. Для того чтобы указать интерпретатору, что аргумент передается по значению, следует использовать ключевое слово `ByVal`. Ниже приведен видоизмененный предыдущий пример, в котором аргумент передается уже по значению.

```
Sub Procedure20()  
    ' Демонстрация передачи по значению  
    Dim i As Integer  
    i = 5  
    ' Передача по значению в другую процедуру  
    Procedure21 i  
    ' Печать результата  
    Debug.Print i  
End Sub  
  
Sub Procedure21(ByVal intInput As Integer)  
    intInput = 12  
End Sub
```

Если теперь запустить процедуру `Procedure20` в окне `Immediate`, на печать будет выведено число 5. Изменения, выполняемые в вызываемой процедуре `Procedure21`, влияют только на копию переменной, но не на ее оригинал.

Присвоение функции типа данных

Как вы уже знаете из главы 3, все переменные имеют некоторый тип данных. Этот же тезис относится и к функциям — в данном случае определяется тип возвращаемого ими значения. Вот пример функции, возвращающей целочисленное значение (тип `Integer`).

```
Function Procedure22(intInput As Integer) As Integer  
    ' умножение аргумента на 10 и возвращение результата  
    Procedure22 = intInput * 10  
End Function
```

Таким образом, синтаксис объявления типа возвращаемого значения ничем не отличается от синтаксиса объявления переменной — за ключевым словом `As` следует название типа данных.

Для значений, возвращаемых функциями, можно использовать любой определенный в языке VBA тип данных. Если тип функции не указан явным образом, возвращаемому значению присваивается тип `Variant`. Как и в случае с обычными переменными, это указывает на то, что тип возвращаемого значения может быть любым.

Обработка ошибок

Во всех процедурах существует потенциальная проблема выхода результата вычислений за пределы диапазона, допустимого для данного типа значений. В нашем случае аргумент определен как Integer, а этот тип не может иметь значение большее, чем 32767. Так что если передать в функцию Procedure22 аргумент 20000, то это приведет к ошибке (рис. 4.6), так как результат вычислений составит 200000, что больше максимально допустимого значения, и возникнет ошибка переполнения.

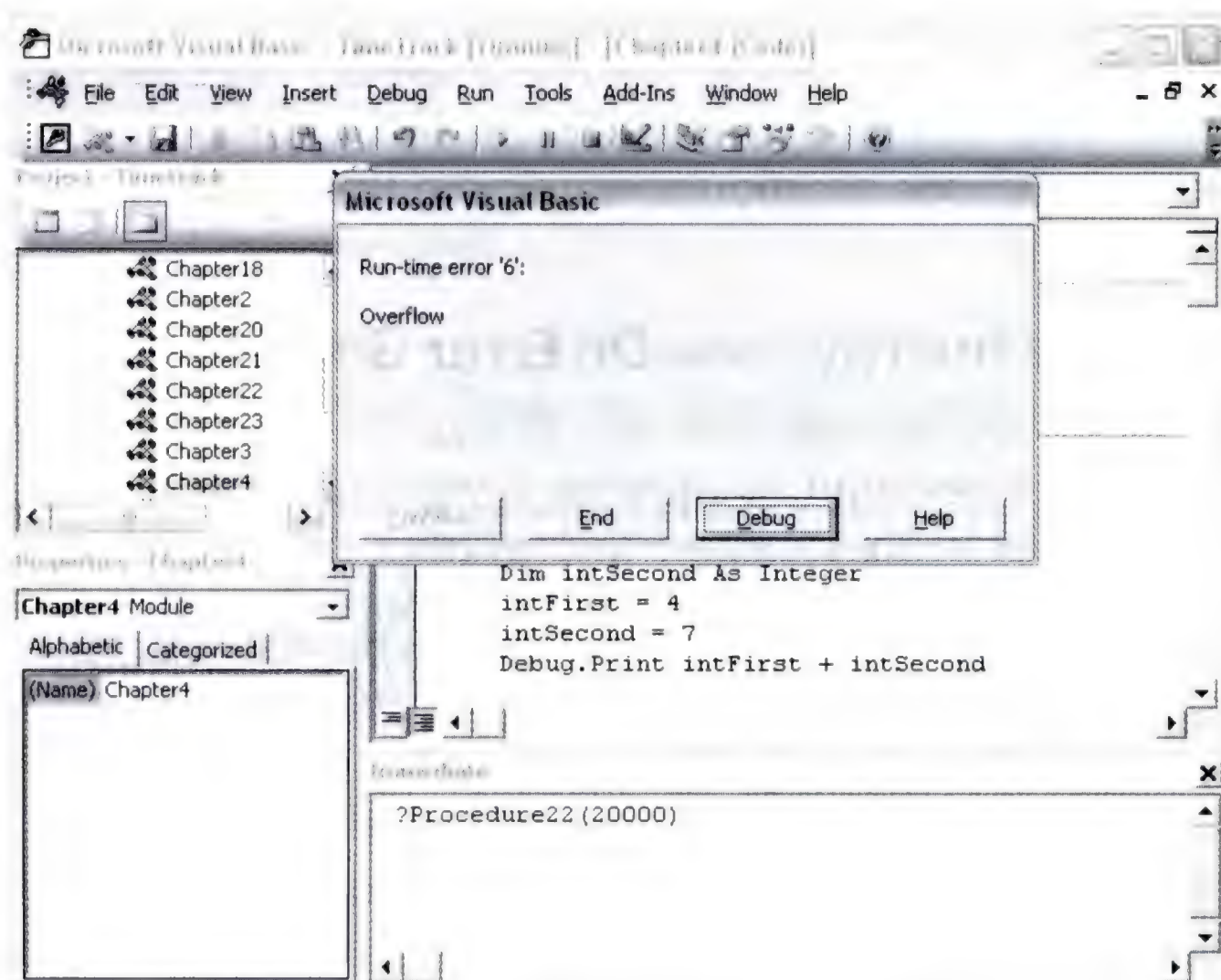


Рис. 4.6. Процедура вызвала ошибку переполнения

Ошибки в программе могут привести к большим проблемам в работе приложения. Это связано с тем, что при возникновении ошибки выполнение программы на VBA прерывается; однако такое поведение принято только по умолчанию. Существуют и другие способы работы с запущенной на выполнение программой, позволяющие выявлять и исправлять ошибки.

Использование инструкции On Error Resume Next

Простейший способ обработки ошибок программы заключается в такой настройке интерпретатора VBA, чтобы он игнорировал возникновение ошибки и переходил к следующему предложению. Этот механизм реализуется с помощью инструкции On Error Resume Next.

```
Function Procedure23(intInput As Integer) As Integer
    ' Умножение аргумента на 10 и возвращение результата
    On Error Resume Next
    Procedure23 = intInput * 10
End Function
```


Теперь если вызвать процедуру `Procedure23` с аргументом, приводящим к переполнению (например, 20000), сообщение об ошибке на экране не появится, а результатом выполнения функции будет нуль. Это происходит потому, что предложение, в котором возникает переполнение, вызывает возникновение ошибки, а инструкция `On Error Resume Next` указывает интерпретатору языка VBA игнорировать ее и переходить к следующему предложению программы — в данном случае к инструкции `End Function`. Функция возвращает нуль по той причине, что он является значением по умолчанию для данного типа данных, а в явном виде переменной ничего присвоено не было.

Обработка ошибок, задаваемая инструкцией `On Error Resume Next`, вступает в силу сразу же после его обработки интерпретатором, и действует до тех пор, пока этот режим обработки ошибок явным образом не будет удален или изменен.

Использование инструкции `On Error Goto`

Несмотря на то, что инструкцию `On Error Resume Next` легко использовать, такой способ реакции на ошибки нельзя назвать самым безопасным. Во многих ситуациях использование инструкции `On Error Goto` будет более выгодным решением — она указывает интерпретатору на выполнение при возникновении ошибок определенного фрагмента программы. Вот пример использования такой обработки ошибок.

```
Function Procedure24(intInput As Integer) As Integer
    ' Умножение аргумента на 10 и возвращение результата
    On Error GoTo HandleErr
    Procedure24 = intInput * 10

ExitHere:
    Exit Function

HandleErr:
    Debug.Print Err.Description
    Resume ExitHere
End Function
```

Эта процедура имеет следующие особенности:

- Инструкция `On Error Goto HandleErr` указывает интерпретатору VBA на то, что при возникновении ошибки выполнение программы следует продолжить со строки, имеющей метку `HandleErr`.
- `ExitHere` и `HandleErr` являются *метками*, помещенными в процедуру в местах, где ее выполнение может продолжиться.
- Инструкция `Exit Function` указывает интерпретатору VBA на то, что следует завершить выполнение функции в данном месте и не рассматривать дальнейшие ее строки.

- `Err.Description` — специальная переменная, содержащая описание последней ошибки.
- `Resume ExitHere` — инструкция по очистке ошибок и продолжению выполнения программы с места, обозначенного меткой `ExitHere`.

Если в данной функции не происходит ошибок, она выводит на печать значение, равное аргументу, умноженному на 10, и переходит к метке `Exit Function`. Если же возникает внештатная ситуация, происходит переход к метке `HandleErr`, на печать выводится информация об ошибке, после чего производится выход из функции. На рис. 4.7 показаны несколько вызовов этой функции в окне `Immediate`.

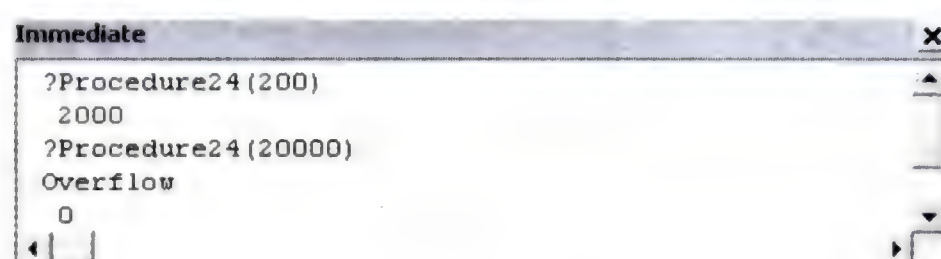


Рис. 4.7. Вызов процедуры с обработкой ошибок

При возникновении ошибок можно использовать еще две инструкции:

- Инструкцию `Resume`, которая указывает интерпретатору VBA на продолжение выполнения программы начиная с той строки, в которой возникла ошибка. Эту инструкцию можно использовать тогда, когда обработчик ошибки способен поправить предусмотренные вероятные внештатные ситуации.
- Инструкцию `Resume Next`, которая указывает интерпретатору VBA на продолжение выполнения программы со строки, следующей за той, в которой возникла ошибка. Эту инструкцию можно использовать в том случае, если обработчик ошибки посчитал, что ошибка является безобидной и ее можно игнорировать.

Отладка программ

Независимо от того, как вы будете настраивать обработку ошибок, от них лучше избавляться. Редактор VBE оснащен некоторыми механизмами, помогающими искать эти ошибки и исправлять их.

Использование режимов Run и Break

После того как процедура была написана, ее необходимо протестировать в среде редактора VBE, прежде чем вставлять в программу, работающую с Access. В первую очередь вы можете просто запустить в редакторе эту процедуру на выполнение и посмотреть, что произойдет в управляемой среде VBE.

Существует несколько способов запуска процедур на выполнение, однако простейшим из них является перемещение курсора внутрь процедуры и нажа-

тие клавиши <F5> (или вызов команды меню Run⇒Run Sub/UserForm). Если процедура работает — вам повезло.

Если же во время выполнения произошла ошибка и причина ее возникновения не очевидна, вам предстоит основательно потрудиться. Для начала нужно идентифицировать источник ошибки, в этом случае можно временно прерывать выполнение процедуры в управляемой среде VBA в определенных точках программы (такой режим называют *режимом прерываний*). Эту задачу можно решить следующим образом:

- установить контрольные точки в отладчике;
- вставить явные инструкции Stop в сам текст программы;
- установить контрольное условие на значение некоторой переменной или выражения;
- нажать комбинацию клавиш <Ctrl+Break> во время выполнения программы (этот метод можно использовать только в крайнем случае, поскольку заранее неизвестно, в какой именно точке находится программа в конкретный момент времени);
- щелкнуть на кнопке Debug (Отладка) в диалоговом окне сообщения об ошибке, если такая возникает;
- щелкнуть на кнопке Break (Останов) на панели инструментов.

После того как выполнение программы приостановлено, у вас есть возможность внести исправление в текст программы, после чего продолжайте тестирование программы, нажав клавишу <F5>. Иногда продолжить выполнение программы требуется не с точки прерывания, а с самого начала. Для этого нужно выполнить команду меню Run⇒Reset *имя_проекта* или щелкнуть на кнопке Reset панели инструментов.

Примечание

Контрольные выражения являются расширенным средством отладки и не будут рассматриваться в этой книге. Данную информацию вы можете почерпнуть из справочной системы VBA, введя в строке поиска фразу "watch expression".

Пошаговое выполнение

Зачастую единственный способ выявления источника проблемы состоит в пошаговом выполнении инструкций программы. Для того чтобы перейти в такой режим работы, нажмите <F8> или выберите в меню команду Debug⇒Step Into. После этого в редакторе будет выделена первая строка программы, как показано на рис. 4.8. Для перехода к следующей инструкции нужно повторно нажать клавишу <F8> и т.д.

В любой момент пошагового процесса вы можете получить информацию о значениях переменных, поместив указатель мыши над названием этой переменной в тексте программы. Текущее значение будет отображено редактором

VBE во всплывающем окне экранной подсказки. Во время этого процесса можно также просмотреть любой фрагмент программы, включая разрешенные ссылки и фрагменты программы, до которых вы еще не дошли в ходе ее выполнения.

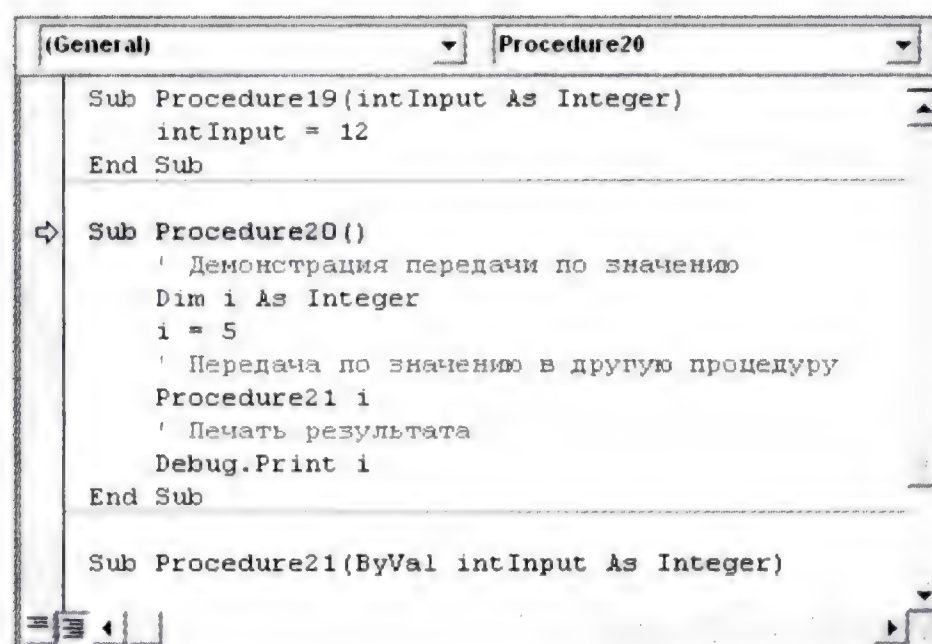


Рис. 4.8. Стрелка и выделение указывают на ту строку, которая в программе будет выполнена следующей в режиме прерываний

При пошаговом выполнении программы вы можете воспользоваться и другими командами.

- Для игнорирования следующей инструкции нажмите клавиши <Shift+F8> (или выполните команду меню **Debug**⇒**Step Over**).
- Для выхода из текущей процедуры нажмите клавиши <Ctrl+Shift+F8> (или выполните команду меню **Debug**⇒**Step Out**). Это не равноценно прерыванию или переустановке процесса выполнения, поскольку в данном случае управление передается вызывающей процедуре (если такая существует).
- Для того чтобы выполнить процедуру от текущего места до инструкции, в которой находится курсор, нажмите клавиши <Ctrl+F8> (или выберите пункт меню **Debug**⇒**Run to Cursor**). Эта команда позволяет безостановочно выполнить несколько инструкций и одновременно контролировать точку останова.

Установка контрольных точек

Еще одним хорошим способом управления местом остановки выполнения программы в режиме прерываний является установка *контрольных точек*. Для этого установите курсор в той строке, в которой вы хотите установить выполнение программы, и нажмите клавишу <F9> (или выберите команду меню **Debug**⇒**Toggle Breakpoint**). При этом на границе окна напротив выбранной строки будет установлен маркер (рис. 4.9).

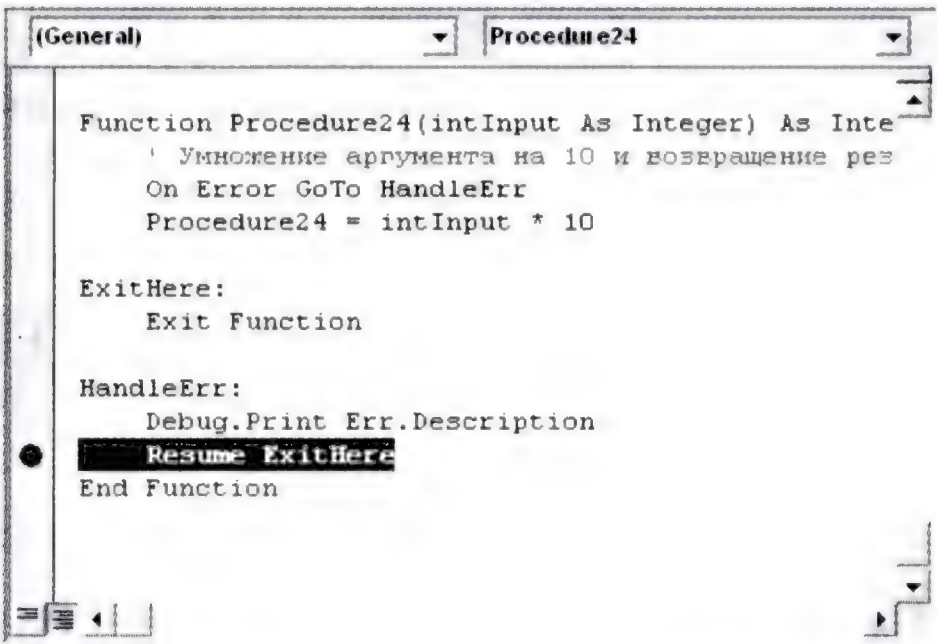


Рис. 4.9. Контрольные точки позволяют более гибко управлять выполнением программы

Запустите процедуру обычным способом. Выполнение прервется в установленной вами контрольной точке. Теперь вы можете использовать пошаговый режим или любой другой метод отладки для точного выявления источника ошибки.

Совет

Для удаления контрольной точки нужно поместить курсор в ее строку и снова нажать <F9>. Альтернативным методом установки и снятия контрольных точек является щелчок на границе окна модуля напротив нужной строки.

Выбор правильных функций VBA

Встроенные функции VBA

VBA — язык высокого уровня, а это значит, что в полной степени овладеть им достаточно сложно. С другой стороны, сочетание VBA и Access позволяет задействовать множество средств автоматизации; и для этого требуется лишь небольшая подготовка.

Существует множество функций, выполняющих десятки задач и вычислений. На стадии обучения не пытайтесь изобретать колесо — используйте уже существующие встроенные функции языка VBA. Познакомившись с ними ближе, вы поймете, насколько они гибки, надежны и удобны для использования.

Подобно всем другим функциям, встроенные функции также возвращают значения; к тому же большинство этих функций имеют обязательные и необязательные аргументы. Благодаря функции IntelliSense добавление в программу встроенных функций не составит большого труда. По мере ввода вызова функции редактор VBE будет отображать списки пригодных для подстановки аргументов. При работе со встроенными функциями VBA учтите следующее:

5

В ЭТОЙ ГЛАВЕ

Встроенные функции языка VBA	79
Функции преобразования типов данных	80
Функции работы с датами	88
Математические и финансовые функции	94
Функции работы со строками	100
Использование функции Format	105
Использование функций семейства Is для беспроблемного выполнения программ	110
Функции взаимодействия с пользователем	111

- Аргументы функций имеют определенный тип данных. Попытка передачи данных неверного типа приведет к ошибке.
- Значения, возвращаемые функциями, также имеют определенный тип данных. Обязательно узнайте о типе данных, возвращаемых функцией, прежде чем вставить ее в текст своей программы, и вы избежите лишних ошибок.

Совет

Самой сложной проблемой, с которой сталкиваются начинающие программисты, является незнание *всех* функций, доступных в языке, и их возможностей. Выйти из трудного положения поможет справочная система редактора VBE. Выберите в меню пункт **Help⇒Справка: Microsoft Visual Basic** — откроется специальная панель справочной системы. В списке тем справки щелкните на пункте **Microsoft Visual Basic Documentation**, после чего откроется список тем документации к языку VBA. Теперь щелкните на подгруппе **Visual Basic Language Reference**, и в открывшемся списке выберите ссылку **Functions (Функции)** или **Statements (Инструкции)**. В результате откроется алфавитный список всех доступных встроенных функций или инструкций языка VBA.

Совет

Вы, наверное, заметили, что термины *функции* и *инструкции* используются в тексте попеременно. С технической точки зрения инструкции лежат в фундаменте языка и были в нем изначально заложены, в то время как функции являются надстройкой, создававшейся позже и постоянно обновляющейся, и используют в своем теле инструкции. Так как программа Access и язык VBA работают с расширенной системой объектов, многие инструкции были заменены функциями. Мы настоятельно рекомендуем, где это возможно, использовать функции, а не инструкции, так как они реализуют самые последние версии алгоритмов. Практически аналогичные им по функциональности инструкции оставлены в системе из соображений обратной совместимости. В то же время следует заметить, что далеко не все инструкции являются устаревшими.

На протяжении всей этой главы (равно как и всей книги) будут использоваться новые функции, пришедшие на смену старым инструкциям (если такие существовали). При этом не обязательно будет указываться, что существуют аналогичные инструкции, поскольку их использование не имеет смысла (если, конечно, вы не будете внедрять приложение, использующее именно их).

Функции преобразования типов данных

В определенный момент вам потребуется импортировать данные из внешних источников или использовать существующие данные не тем способом, для которого они изначально создавались. В такой ситуации практически всегда возникает необходимость в преобразовании данных из одного типа в другой. Для этих целей в языке VBA существует множество специальных

функций. При их использовании помните, что вы не изменяете тип исходных данных, а преобразуете его в новые данные нового типа.

➔ Более глубокий обзор типов данных, используемых в языке VBA, приведен в разделе “Типы данных в VBA” главы 3.

Для того чтобы получить полный список функций преобразования типов данных, откройте справочную панель редактора VBE и введите в строке поиска фразу “Type Conversion Functions”. В настоящем же разделе мы подробно остановимся только на самых распространенных из этих функций:

- **CBool**. Функция преобразования значения в булев тип.
- **CByte**. Функция преобразования значения в тип **Byte**.
- **CDate**. Функция преобразования значения в тип даты.
- **CInt**. Функция преобразования значения в целочисленный тип.
- **CStr**. Функция преобразования значения в строковый тип.
- **CVar**. Функция преобразования значения в тип **Variant**.

Совет

Некоторые устаревшие инструкции преобразования, такие как **Str**, продолжают поддерживаться в современных версиях VBA. Все современные функции преобразования начинаются с буквы **C**. Рекомендуется использовать именно их, поскольку, в отличие от устаревших инструкций, они учитывают системные настройки даты, времени и чисел.

Все функции преобразования используют одинаковый простой синтаксис, такой как

Cbool (*переменная*)

где *переменная* — имя переменной, константы или выражение, преобразовываемой в другой тип данных. Преобразованное значение следует присвоить переменной определенного типа, после чего ее можно использовать в любом месте программы; при этом с исходным значением ничего не происходит. В то же время следует заметить, что не все типы данных поддаются преобразованию во все остальные типы. В следующих разделах, посвященных конкретным функциям, этот вопрос будет рассмотрен отдельно.

Преобразование в булев тип данных

Для преобразования в булев тип данных используется функция **CBool**. Аргумент *переменная* является обязательным и может быть строковым или числовым. Вы можете подумать, что преобразовываемое значение должно обязательно, явно или неявно, быть равным 0 или –1, но это далеко не так. Преобразовываемое значение может быть любым числом или строкой, которую можно интерпретировать как число.

Когда значение числа или строки равно нулю, функция возвращает значение `False`; когда оно отлично от нуля — `True`. В примере все следующие выражения вернут значение `True`, поскольку аргумент может интерпретироваться как число, отличное от нуля:

```
CBool("1")
CBool(1+0)
CBool(2)
CBool(-300)
```

С другой стороны, оба следующих выражения вернут значение `False`, так как аргумент может интерпретироваться только как число 0:

```
CBool(0)
CBool("0")
```

Функция `CBool` не может работать абсолютно со всеми символами — интерпретатор VBA должен иметь возможность интерпретировать строку как некоторое числовое значение или выражение. Этой строкой могут быть числа и выражения, содержащиеся в формате строки, но не буквы. На рис. 5.1 показан пример, в котором при попытке использования в качестве аргумента функции `CBool` буквенной строки интерпретатор выдал сообщение об ошибке, поскольку слово `"one"` (один) он не смог интерпретировать как число 1.

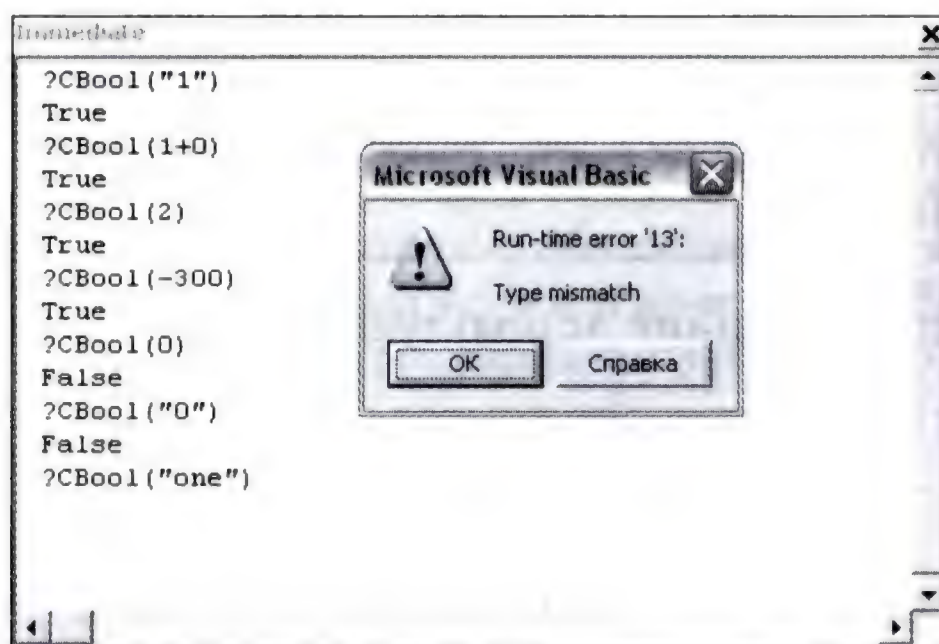


Рис. 5.1. Функция `CBool` вернула сообщение об ошибке, поскольку не смогла интерпретировать аргумент как число

Совет

Функция `CBool` преобразовывает любое значение и выражение, интерпретируемое как ноль, в значение `False`, а отличное от нуля — в `True`. Впоследствии, при обратном преобразовании значение `False` будет преобразовано в число 0, а `True` — в значение -1.

Преобразование в тип данных `Byte`

В переменных типа `Byte` можно хранить целочисленные значения в диапазоне от 0 до 255. Аргумент *переменная* является обязательным и может

быть строкового или числового типа. Функция `CByte` производит преобразование любых чисел из диапазона от 0 до 255 или любых строк, которые можно интерпретировать как значение или выражение, результат которого лежит в этом диапазоне. Следующие три инструкции вернут значения 0, 255 и 1 соответственно:

```
CByte(0)
CByte(255)
CByte("1")
```

Когда преобразуемое значение интерпретировано как число, выходящее за пределы допустимого диапазона, интерпретатор выдает сообщение об *ошибке переполнения* (рис. 5.2).

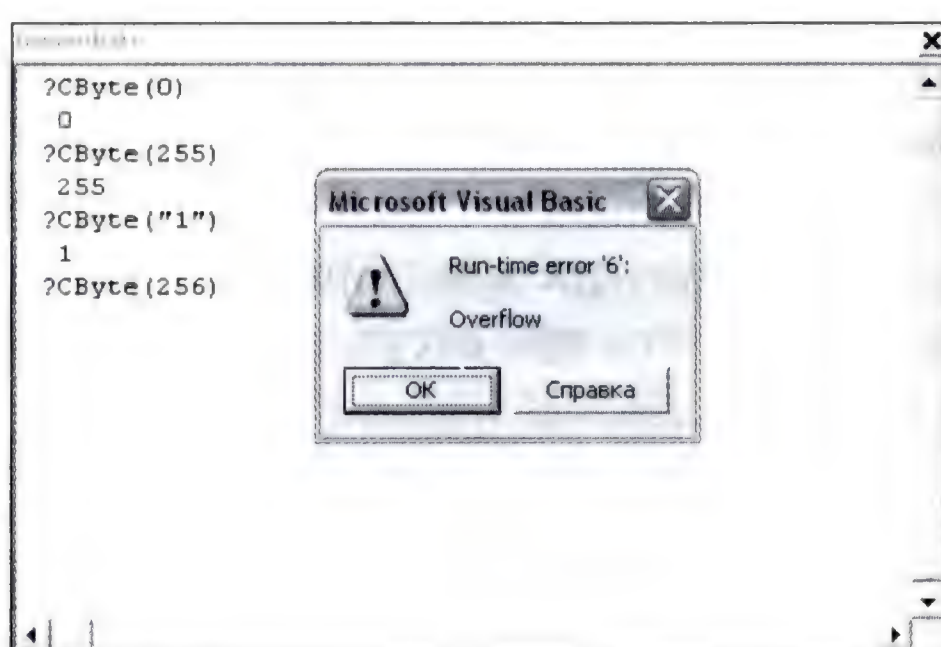


Рис. 5.2. Передача функции `CByte` значения, выходящего за пределы допустимого диапазона, вызывает сообщение об ошибке

Совет

Перед преобразованием числа с плавающей точкой в тип `Byte` функция `CByte` производит его округление до ближайшего целого числа. Если округленное значение попадает в диапазон от 0 до 255, функция его преобразовывает.

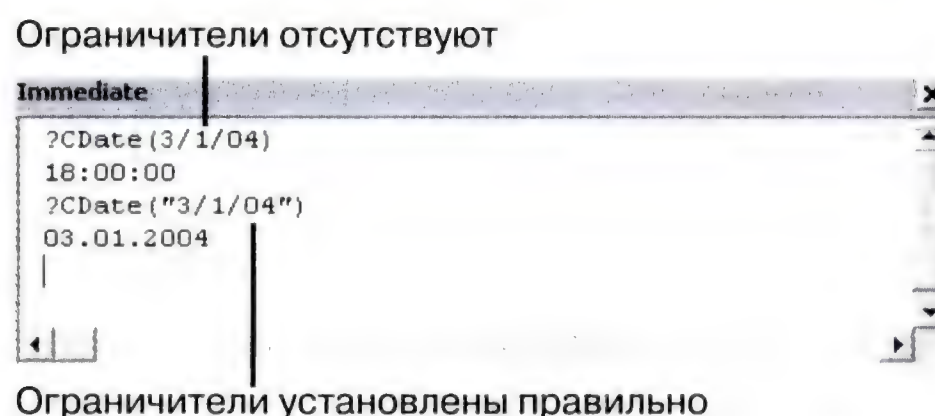
Преобразование в тип даты

Функция `CDate` преобразовывает значение в тип даты. Аргумент *переменная* является обязательным и может иметь строковый или числовой тип, а также представлять собой допустимый результат выражения. Функция `CDate` использует региональные настройки операционной системы для определения порядка следования трех компонентов даты: дня, месяца и года. При преобразовании данных в тип даты вам помогут нижеследующие ориентиры.

- Функция `CDate` преобразовывает целую часть числа в дату, которая стоит на данное количество дней от 31 декабря 1899 года. Дробная часть числа преобразовывается во время (где 24 часа суток умножаются на дробную часть), при этом значение 0.01 соответствует 14 минутам и 24 секундам (864 секундам), прошедшим после полуночи.

- Функция `CDate` принимает наряду с числовыми и строковые значения. К примеру, она может корректно интерпретировать такие строковые значения, как “3/1/04”, “5 марта 2004”, “3 мар 2004” и число 38047 (соответствующее дате 1 марта 2004 года).
- Тип `Date` поддерживает даты от 1 января 100 года до 31 декабря 9999 года; любой аргумент, выходящий за эти пределы, вызывает ошибку.
- Сокращенное указание года (в виде двух цифр) приводит к преобразованию в дату XXI века, если это число меньше 30, в противном случае преобразование производится в дату XX века. Если в аргументе вообще отсутствует год, подставляется текущий год, установленный в системе.

На рис. 5.3 показан неожиданный результат выполнения функции `CDate`. В данном случае аргумент `3/1/04` был интерпретирован как числовое выражение, результат которого (равный `0.75`) был преобразован во время суток (`18:00:00` — это соответствует трем четвертям суток, прошедших после полуночи) (обратите внимание, что при отсутствии круглых скобок вычисления выполняются слева направо).



Ограничители установлены правильно

Рис. 5.3. Правильно ограничивайте значение аргумента

Причиной ошибки послужило отсутствие кавычек, ограничивающих строковое значение. Если их установить, преобразование в дату выполнится и даст ожидаемый результат.

Преобразование в целочисленный тип данных

Функция `CInt` преобразовывает значение в целочисленный тип данных. Аргумент *переменная* является обязательным и может представлять собой любую переменную, константу или выражение в диапазоне от `-32678` до `32767`. Когда дробная часть числа равна `0.5`, округление производится в сторону ближайшего четного целого числа. К примеру, число `0.5` будет округлено до нуля, а `1.5` — до двух. В обоих следующих примерах результатом функции будет число `1000`:

```
CInt(1000)
CInt("1000")
```


Если в аргументе содержится нечто отличное от числового значения или выражения, функция вызывает сообщение об ошибке несовместимости типов. Если аргумент выходит за пределы допустимого числового диапазона, генерируется сообщение об ошибке переполнения.

Совет

Функция `CInt` является более гибкой, чем `Val`, поскольку при распознавании разделителя дробной части использует региональные настройки.

Преобразование в строковый тип данных

Функция `CStr` преобразовывает практически любое значение в строковый тип данных. Аргумент *переменная* является обязательным и может представлять собой любую переменную, константу или выражение, которые можно интерпретировать как строку. Несмотря на свою универсальность, функция может привести к некоторым неожиданным результатам:

- Неинициализированная числовая переменная будет преобразована в строку "0".
- Неинициализированная переменная типа `Date` будет преобразована в строку "12:00:00".

Совет

Под *инициализацией* понимается первое присвоение переменной некоторого значения. Другими словами, переменная остается *неинициализированной* между ее объявлением и первым оператором присвоения ей значения.

Преобразование в тип данных Variant

В разделе “Типы данных в VBA” главы 3 тип `Variant` был представлен как самый универсальный, поскольку позволяет хранить практически любое значение. С помощью функции `CVar` можно преобразовать практически любое числовое и строковое значение в тип `Variant`. Числовые данные ограничены диапазоном, установленным для типа `Double`; для нечисловых же значений ограничений не существует.

Предупреждение

Лучше всего использовать функцию `CVar` в случаях, когда тип данных не играет роли (а такая ситуация встречается крайне редко).

Преобразование нулевых значений

Лишь немногие функции могут работать с нулевыми значениями, поэтому когда переменная равна специальному значению `Null`, это может стать источником ошибки, даже несмотря на то, что данное значение в среде VBA является вполне корректным. К примеру, простая инструкция


```
varResults = value1 + value2
```

вызовет сообщение об ошибке, если какое-либо из слагаемых равно Null.

Этой ошибки можно избежать, если с помощью функции Nz преобразовать константу Null в значение 0 или пустую строку (" "). После обрамления обоих слагаемых (см. предыдущий пример) функцией Nz при наличии хотя бы одного значения Null ошибки уже не будет:

```
varResults = Nz(value1) + Nz(value2)
```

Функция Nz имеет следующий синтаксис:

```
Nz(значение, [значение_если_Null])
```

где *значение* — это данные любого типа, представляющие значение или выражение, а *значение_если_Null* — то значение, которое функция будет возвращать, если значением первого аргумента будет Null. Если второй необязательный аргумент опустить, функция возвращает, в зависимости от типа данных, нуль или строку нулевой длины.

Пример преобразования

В следующем примере пользователь вводит дату, над которой затем выполняется простая арифметическая операция. Предположим, что пользователю требуется узнать, на сколько дней текущая дата отстоит от даты начала проекта, которую он вводит в диалоговом окне.

Для начала запустим редактор VBE, нажав в Access клавиши <Alt+F11>. После этого выполним следующие действия:

1. Откроем новый пустой модуль, выбрав в меню команду Insert⇒Module.
2. В окне модуля введем следующую процедуру:

```
Public Sub GetDate()
    Dim varDate As Variant
    varDate = InputBox("Пожалуйста, введите дату")
    varDate = CDate(varDate)
    'вывод в окне количества дней разницы
    MsgBox Now - varDate
End Sub
```

➔ Процесс ввода процедуры подробно описан в разделе “Ввод и запуск программ на языке VBA” главы 2.

3. Переместите курсор в тело процедуры и нажмите клавишу <F5>, чтобы запустить ее на выполнение.
4. Когда откроется диалоговое окно, показанное на рис. 5.4, введите корректную дату в текстовое поле и щелкните на кнопке ОК.
5. Когда интерпретатор VBA отобразит сообщение об ошибке несоответствия типов, щелкните на кнопке Debug (отладка) и вернитесь к процедуре. На рис. 5.5 показана строка, в которой произошла ошибка (в окне она выделена).
6. На стандартной панели инструментов щелкните на кнопке Reset.

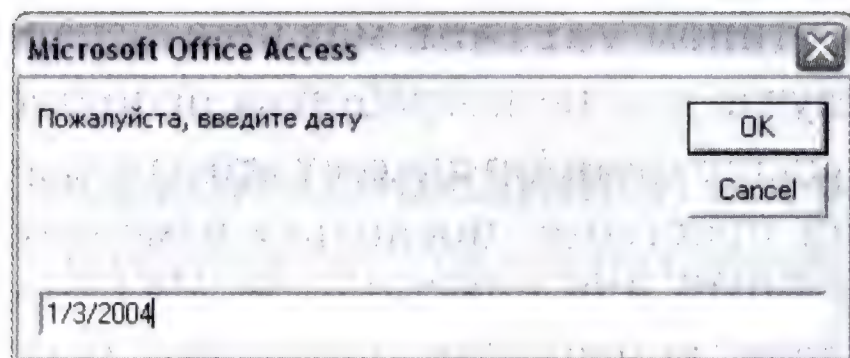


Рис. 5.4. Введите в текстовое поле корректную дату

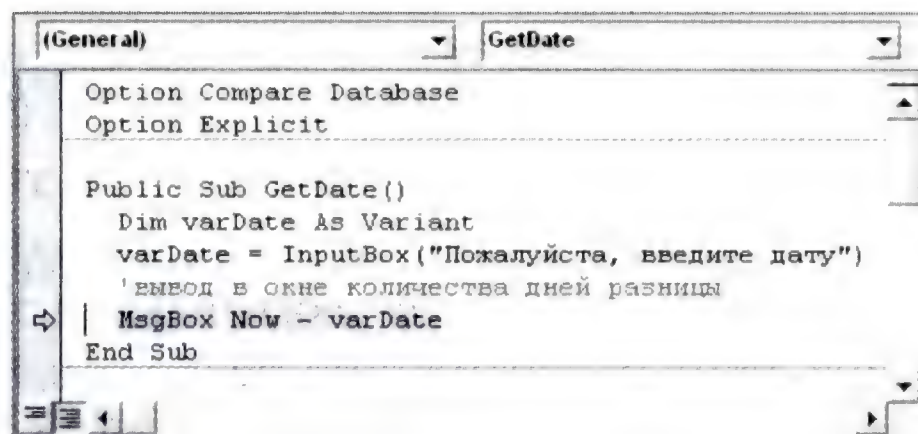


Рис. 5.5. Инструкция MsgBox привела к ошибке

Примечание

В некоторых примерах используется встроенная функция ввода данных языка VBA — InputBox. Подобно функции MsgBox, вы встретите ее в нескольких местах книги. В своей упрощенной форме эти функции требуют передачи всего одного строкового аргумента: в первом случае это сообщение, отображаемое в диалоговом окне ввода значения и напоминающее пользователю, что именно от него требуется; во втором случае — текст, отображаемый в окне сообщения. Функция ввода возвращает значение подтипа Variant String. В настоящем контексте этот подтип является одним из типов данных, совместимым с типом Variant.

Вся проблема заключена в операции вычитания из результата функции Now (она возвращает текущую дату) даты, введенной пользователем. Данная математическая операция не может быть выполнена системой по той причине, что введенные пользователем данные имеют тип Variant, а функция Now возвращает данные типа Date. Для выполнения операции требуется, чтобы оба операнда имели один и тот же тип данных. Одним из способов решения проблемы является преобразование введенного пользователем значения в тип даты.

1. Вставьте между строками InputBox и MsgBox пустую строку и введите в нее следующее выражение:
`varDate = CDate(varDate)`
2. Установив курсор внутри процедуры, нажмите клавишу <F5>.
3. В диалоговом окне введите ту же дату и щелкните на кнопке OK.
4. На этот раз в результате выполнения программы откроется окно сообщения, в котором будет отображено количество дней разницы между текущей и введенной датами (рис. 5.6).
5. Щелкните на кнопке OK и при желании сохраните модуль под именем Chapter5.



Рис. 5.6. После преобразования введенных данных в тип даты арифметическая операция ошибок не вызывает

После того как строка даты изменила свой тип с Variant на Date, интерпретатор VBA смог вычислить количество дней разницы между этими двумя

датами. Существует еще один, более простой способ решения этой проблемы. Вместо объявления переменной `varDate` с типом `Variant` можно изначально установить для нее тип `Date`. В этом случае вам не потребуется использовать функцию преобразования, поскольку интерпретатор сделает это за вас.

Решения не всегда оказываются такими простыми, поскольку изменение типа данных переменной редко бывает удобным или практичным. На самом деле вряд ли вы станете этим заниматься, если, конечно, не определите (как в приведенном примере), что причиной ошибки стал неправильно установленный изначально тип данных переменной `varDate`.

Функции работы с датами

Работа с датами может оказаться сложной, если вы не знаете основ языка VBA. В то же время, если вы используете даты в выражениях Access (в запросах, вычисляемых столбцах и т.п.) и это не вызывает проблем, вы можете применить свой опыт и при работе с VBA. В этом разделе будет рассмотрен ряд функций VBA, работающих с датами.

Предупреждение

Все функции работы с датами имеют те же ограничения, что сам тип `Date`. Первая распознаваемая дата — 1 января 100 года, последняя — 31 декабря 9999 года. Любая дата, не попадающая в этот диапазон, вызывает ошибку.

Возвращение даты

Самая простая функция работы с датами — функция `Date`. Она имеет предельно простой синтаксис:

`Date`

и возвращает значение типа `Variant Date`, равное текущей системной дате. (Функция `Date$` возвращает значение строкового типа.)

Совет

Функция `Date` возвращает только текущую системную дату. Для того чтобы получить и дату и время, используйте функцию `Now`.

Для переустановки текущей системной даты используйте инструкцию `Date`, имеющую следующий синтаксис:

`Date = новая_дата`

где `новая_дата` может иметь тип `String`, `Date` или `Variant Date`. К примеру, инструкция

`Date = "1 марта 2004"`

устанавливает системную дату на первое марта 2004 года. В то же время не рекомендуется использовать данный метод, если на то не существует особых

причин. Изменение системной даты может иметь далекоидущие непредвиденные последствия.

Операции сложения и вычитания дат

Функция `DateAdd` используется для добавления и вычитания конкретного числа периодов времени из заданной даты. К примеру, можно вычислить дату, которая отстоит на 10 дней или 10 месяцев от заданной, в прошлом или будущем. Также можно вернуть дату и время, отстоящие от заданных на 36 часов.

Функция в общем виде имеет следующий синтаксис:

```
DateAdd(интервал, количество, дата)
```

где *интервал* является строковым значением или выражением, указывающим на тип периода (день, неделя или месяц), который следует добавлять или вычитать. В табл. 5.1 перечислены предопределенные значения интервала, существующие в языке VBA. Аргумент *количество* является числовым значением, определяющим количество *интервалов*, которые следует добавить или вычесть из *даты*, которая, в свою очередь, имеет тип `Date Variant`.

Таблица 5.1. Значения аргумента *интервал*

Строковое значение	Описание
yyyy	Год
q	Квартал
m	Месяц
y	День года
d	День
w	День недели
ww	Неделя
h	Час
m	Минута
s	Секунда

Отрицательное значение аргумента *количество* означает вычитание нужного числа *интервалов* из *даты*, а положительное — добавление. Если значение аргумента *количество* дробное, перед реальными вычислениями функции оно округляется до ближайшего целого числа.

Вычисление разницы между двумя датами

Для вычисления количества временных интервалов между двумя датами используется функция `DateDiff`. Эта функция возвращает значение подтипа `Variant Long` и имеет следующий синтаксис:

DateDiff(*интервал*, *дата1*, *дата2* [, *первый_день_недели* [, *первая_неделя_года*]])

где *интервал* — строковое значение или выражение, указывающее на тип периодов, в которых будет выводиться результат вычитания из *даты1* *даты2* (см. табл. 5.1). Два необязательных аргумента *первый_день_недели* и *первая_неделя_года* являются числовыми константами, определяющими первый день недели и года соответственно (табл. 5.2 и 5.3). Если эти аргументы опущены, первым днем недели считается воскресенье, а первой неделей года та, на которую приходится 1 января.

Таблица 5.2. Константы первого дня недели

Константа	Описание	Числовое значение
vbSunday	Воскресенье (принято по умолчанию)	1
vbMonday	Понедельник	2
vbTuesday	Вторник	3
vbWednesday	Среда	4
vbThursday	Четверг	5
vbFriday	Пятница	6
vbSaturday	Суббота	7

Таблица 5.3. Константы первой недели года

Константа	Описание	Числовое значение
vbFirstJan1	Неделя с первым января (принято по умолчанию)	1
vbFirstFourDays	Первая неделя, содержащая минимум 4 дня нового года	2
vbFirstFullWeek	Первая полная неделя нового года	3

Эта функция не всегда возвращает ожидаемые значения, например:

- Если *дата2* оказывается раньше *даты1*, функция возвращает отрицательное значение.
- Даже несмотря на то, что 31 декабря старого года и 1 января нового отделяет всего один день, функция DateDiff считает, что их разделяет целый год. Аналогичная ситуация возникает и при вычислении разницы в кварталах и месяцах.

На рис. 5.7 показаны примеры этих неадекватных результатов.

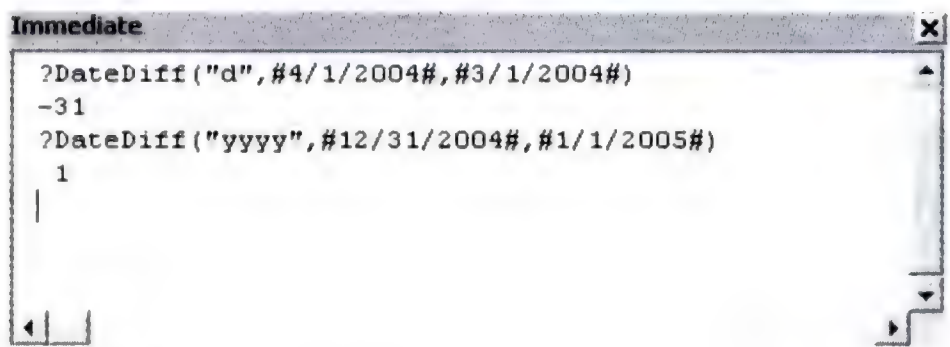


Рис. 5.7. Учтите не всегда адекватные вычисления функции DateDiff

Примечание

Константы дат в примерах на рис. 5.7 обрамлены знаками решетки (#). Этот ограничитель является общепринятым для значений типа Date, так же как кавычки для строковых значений. Если не использовать эти ограничители, то Access или функции VBA могут неверно интерпретировать тип данных и выдать сообщение об ошибке или вернуть некорректный результат вычислений.

Извлечение компонентов даты

С помощью функции DatePart из даты можно извлечь отдельный ее компонент. К примеру, следующие функции возвращают результаты 4, 1 и 2004 соответственно¹:

```
DatePart("m", #1/4/2004#)
DatePart("d", #1/4/2004#)
DatePart("yyyy", #1/4/2004#)
```

Эта функция возвращает значение подтипа Integer Variant и имеет следующий синтаксис:

```
DatePart(интервал, дата[, первый_день_недели[, первая_неделя_года]])
```

где *интервал* — это строковое значение или выражение, указывающее на компоненту (см. табл. 5.1), интересующую пользователя в *дате*, в свою очередь имеющую тип Date Variant. Два необязательных аргумента, *первый_день_недели* и *первая_неделя_года*, являются числовыми константами, определяющими первый день недели и года соответственно (см. табл. 5.2 и 5.3). Если эти аргументы опустить, первым днем недели будет считаться воскресенье, а первой неделей года та неделя, на которую приходится 1 января.

Создание даты из отдельных компонентов

В предыдущем разделе вы узнали, как с помощью функции DatePart извлекать из даты отдельные ее компоненты. Функция DateSerial создана для

¹ Здесь и далее предполагается, что в операционной системе установлены региональные настройки, принятые для России по умолчанию. — *Примеч. ред.*

обратной цели — сборки отдельных компонентов в значение типа даты — Variant Date. Эта функция имеет следующий синтаксис:

DateSerial(*год, месяц, день*)

Все аргументы функции имеют целочисленный тип и представляют одноименные компоненты даты. В то же время существуют отдельные ограничения значений этих аргументов:

- все аргументы являются обязательными;
- *год* может принимать числовые значения от 100 до 9999 включительно;
- *месяц* может принимать числовые значения от 1 до 12 включительно;
- *день* может принимать числовые значения от 1 до 31 включительно.

Совет

Как и при работе с остальными функциями, при сокращенном (двузначном) указании года подразумевается, что числа от 0 до 29 соответствуют XXI веку, а от 30 до 99 — XX. В целях безопасности вычислений рекомендуется всегда использовать полный, четырехзначный формат года, а не отдавать на откуп интерпретатору VBA принятие решения по этому поводу.

Если любой из аргументов выпадает за пределы допустимого диапазона значений, функция DateSerial применит кратное значение к следующему по старшинству периоду (к примеру, лишние 12 месяцев прибавят 1 год и т.п.). В частности, функция

DateSerial(2004, 15, 3)

вернет в качестве результата дату 3 марта 2005 года, т.е. прибавит к 2004 году 12 месяцев и получит год 2005-й, вычтя то же количество из аргумента *месяц*. Аналогично, функция

DateSerial(2004, 4, 45)

вернет в качестве результата дату 15 мая 2005 года, увеличив значение месяца на единицу за счет вычитания 30 дней из аргумента *день* (так как в апреле 30 дней).

Создание даты из строкового выражения

Функция DateValue языка VBA позволяет преобразовать в дату строковые значения. Она возвращает значение подтипа Variant Date и имеет следующий синтаксис:

DateValue(*строковое_выражение*)

где *строковое_выражение* — обязательный аргумент, использующий настройки короткой даты операционной системы. В дополнение, разделители отдельных компонентов даты должны соответствовать тем, которые установлены в региональных настройках системы. Информация о времени игнорируется; если же функции передается только информация о времени, она возвращает ошибку. Все три описанные ниже функции вернут значение, соответствующее 1 марта 2004 года:


```
DateValue("1/3/2004")
DateValue("1 марта 2004")
DateValue("1 мар 2004")
```

Совет

Функции TimeSerial и TimeValue работают аналогично DateSerial и DateValue, однако используют компоненты и строковые представления времени, а не дат.

Извлечение заданного компонента даты и времени

Некоторые функции (табл. 5.4) возвращают заданный компонент даты или времени. Их легко использовать, так как единственным передаваемым им аргументом является дата.

Таблица 5.4. Функции извлечения компонентов даты

Функция	Результат
Day(<i>дата</i>)	Целое число в диапазоне от 1 до 31, соответствующее дню месяца
Hour(<i>дата</i>)	Целое число в диапазоне от 0 до 23, соответствующее часу дня
Minute(<i>дата</i>)	Целое число в диапазоне от 0 до 59, соответствующее минуте
Second(<i>дата</i>)	Целое число в диапазоне от 0 до 59, соответствующее секунде
Month(<i>дата</i>)	Целое число в диапазоне от 1 до 12, соответствующее месяцу
Year(<i>дата</i>)	Целое число, соответствующее году
Day(<i>дата</i>)	Целое число в диапазоне от 1 до 31, соответствующее дню месяца

Пример использования функции дат

Приведенная в примере функция позволит узнать, сколько дней выделено на реализацию проекта. С помощью описанной ниже процедуры введите плановые даты начала и конца проекта и немедленно получите результат.

```
Public Function GetDays() As Integer
    Dim dtmStart As Date
    Dim dtmEnd As Date
    Dim varDays As Variant
    dtmStart = InputBox("Введите дату начала")
    dtmEnd = InputBox("Введите дату завершения")
    varDays = DateDiff("d", dtmStart, dtmEnd)
    GetDays = varDays
End Function
```

1. Введите вышеуказанную процедуру в модуль Chapter5 базы данных примера.
2. В окне Immediate введите инструкцию ?GetDays и нажмите клавишу <Enter>.
3. Когда откроется первое диалоговое окно, введите в него дату 1/3/2004 и щелкните на кнопке ОК.

4. Во втором открывшемся окне введите дату **1/4/2003** и также щелкните на кнопке **ОК**.

На рис. 5.8 показан полученный результат — на выполнение проекта вам отведен 31 день.

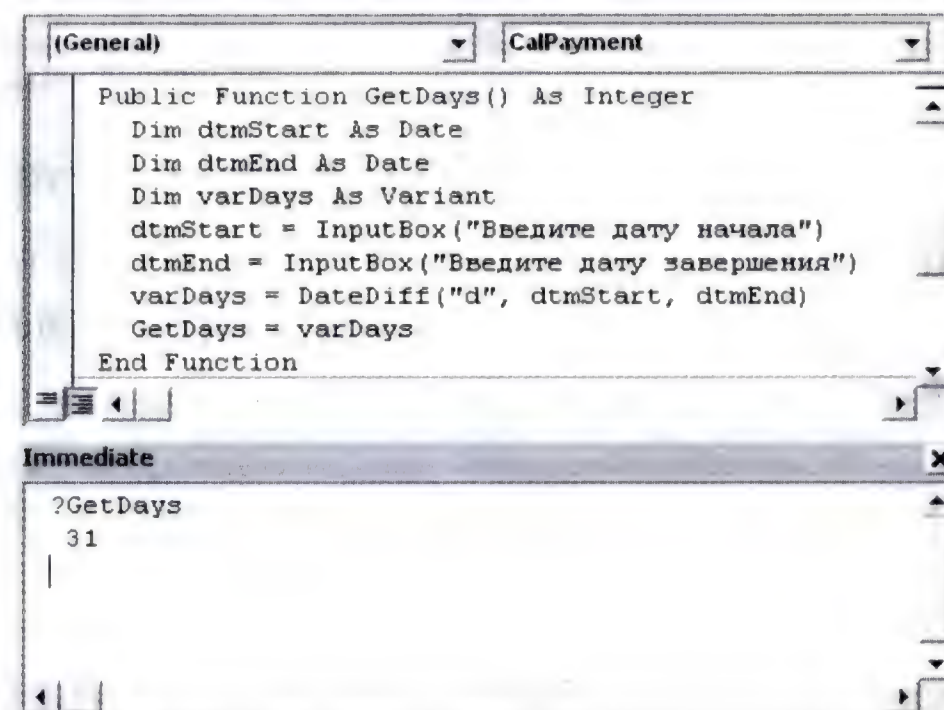


Рис. 5.8. Определение количества дней, отведенных на реализацию проекта

Математические и финансовые функции

Все множество математических, статистических и финансовых функций практически невозможно охватить в одном коротком разделе. Большей частью все эти функции используют операторы и операнды для работы с числовыми данными. В настоящем разделе мы не будем подробно останавливаться на каждой из функций, однако постараемся провести как можно более полный обзор наиболее часто используемых.

Функция Abs

Функция `Abs` возвращает абсолютное значение передаваемого числового аргумента и имеет следующий синтаксис:

`Abs (число)`

где *число* — любое допустимое числовое выражение или константа. Возвращаемое значение является числом того же типа, но уже без знака. К примеру, в функции

`Abs(intValue)`

переменная `intValue` имеет тип `Integer`; если переменная имеет значение `-3`, то функция вернет `3`.

Функция Int

Функция `Int` возвращает целую часть передаваемого числового аргумента и имеет следующий синтаксис:

`Int (число)`

где *число* — любое допустимое числовое выражение или константа. Эта функция не выполняет округлений — она просто удаляет дробную часть *числа*, если такая существует. К примеру, функция

`Int (10.9)`

вернет значение 10, а не 11. Если аргумент *число* окажется отрицательным, будет возвращено ближайшее целое число, которое меньше или равно аргументу. К примеру, функция

`Int (-10.9)`

вернет значение -11.

При преобразовании значений вместо функции `Int` рекомендуется использовать `CInt`, однако помните, что эти функции не являются абсолютно взаимозаменяемыми. Первая из них не может преобразовывать тип передаваемого *числа*. Несмотря на то, что функция `CInt` является более предпочтительным выбором, она может вернуть другие результаты. Оцените стоящую перед вами задачу и сделайте свой выбор согласно обстоятельствам.

Функция Rnd

Функция `Rnd` используется для получения случайного числа. Ее единственным и к тому же необязательным аргументом может быть любое допустимое числовое выражение. Эта функция имеет следующий синтаксис:

`Rnd [ (распределение) ]`

где *распределение* определяет диапазон возвращаемых случайных чисел.

- Если *распределение* меньше нуля, функция возвращает одно и то же случайное число.
- Если *распределение* больше нуля, функция возвращает следующее значение в последовательности случайных чисел.
- Если *распределение* равно нулю, функция возвращает наиболее часто генерируемое случайное число.
- Если аргумент *распределение* опущен, функция возвращает следующее значение в последовательности случайных чисел, как и заложено изначально в ее конструкции.

Совет

Иногда приходится работать со значениями, которые не генерируют случайные числа так, как того хотелось бы. Для примера предположим, что вы работаете со значениями, которые меньше нуля, но вам совершенно не нужно повторение одного и того же случайного числа. В данном случае вы можете использовать инструкцию `Randomize`, которая переустанавливает внутреннее *распределение* таким образом, чтобы функция `Rnd` возвращала неповторяющиеся числа, которые будут выглядеть как случайные.

Пример математической функции

В первом примере главы мы использовали функцию `CDate` для преобразования значения типа `Variant` в значение типа даты. Как вы помните, результат представлял из себя дробное число. Если в это равенство добавить функцию `Int`, результат станет целочисленным. Для этого:

1. Вернитесь к модулю `Chapter5` и найдите процедуру `DetDate()`.
2. Закомментируйте инструкцию `MsgBox`, выделив эту строку и щелкнув на кнопке **Comment Block** панели инструментов; или же просто введите символ апострофа (`'`) в начале этой строки.
3. Введите следующую строку программы
`MsgBox Int(Now - varDate)`
4. Установив курсор в любом месте внутри этой процедуры, нажмите клавишу `<F5>`.
5. В открывшемся диалоговом окне введите дату **1/3/2004** и щелкните на кнопке **ОК**. На этот раз в диалоговом окне результата будет отображено целое число количества прошедших дней с момента начала проекта (рис. 5.9). Щелкните на кнопке **ОК**.

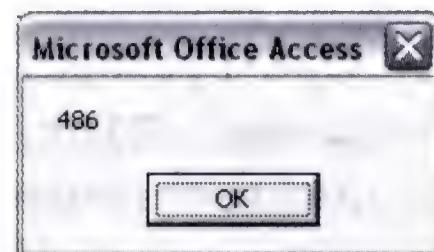


Рис. 5.9. Дополнение инструкции `MsgBox` функцией `Int` сделало отображаемое сообщение более понятным

Функция `Ddb`

Функция `Ddb` рассчитывает амортизацию предмета для заданного промежутка времени при помощи метода двойного уменьшения остатка или другого метода. Эта функция имеет следующий синтаксис:

`DDB(стоимость, остаток, время_жизни, период[, фактор])`

где *стоимость* — первоначальная стоимость объекта основных фондов; *остаток* — полезная стоимость объекта по окончании срока полезного использования; *время_жизни* — продолжительность полезного использования предмета; *период* — период амортизации объекта; *фактор* — необязательный аргумент, определяющий метод расчета амортизации (по умолчанию принято значение 2, соответствующее методу двойного уменьшения остатка²). Первые четыре аргумента имеют тип `Double`, последний — `Variant`.

Функция `Fv`

Функция `Fv` возвращает значение типа `Double`, оценивающее суммарную величину будущих выплат, исходя из количества периодов внесения платежей, а также предположения о фиксированных суммах платежей и постоянной учетной ставке. Функция имеет следующий синтаксис:

`FV(ставка, число_периодов, платеж[, единовременный_платеж[, тип]])`

² В документации не описаны другие возможные значения этого аргумента. — *Примеч. ред.*

где *ставка* — процентная ставка за период; *число_периодов* — количество периодов внесения платежей; *платеж* — величина выплаты, которая должна производиться каждый период; *единовременный_платеж* — необязательный аргумент, равный величине единовременно выплачиваемой суммы для серии будущих платежей; *тип* — необязательный аргумент, задающий время внесения платежей (начало или конец периода). Первые три аргумента имеют тип `Double`, последние два — `Variant`.

Функция IPmt

Функция `IPmt` возвращает значение типа `Double`, соответствующее величине выплаты по ссуде за указанный период на основе постоянных выплат и постоянной учетной ставки. Эта функция имеет следующий синтаксис:

```
IPmt(ставка, период, число_периодов, единовременный_платеж[,  
    будущий_платеж[, тип]])
```

где *ставка* — процентная ставка за период; *период* задает период выплаты в интервале от единицы до *число_периодов* (общего количества периодов); *единовременный_платеж* — величина единовременно выплачиваемой суммы или текущего значения для серии будущих выплат или поступлений; *будущий_платеж* — необязательный аргумент, равный будущему значению требуемого остатка после последней выплаты; *тип* — необязательный аргумент, задающий время внесения платежей (начало или конец периода). Первые четыре аргумента имеют тип `Double`, последние два — `Variant`.

Функция NPer

Функция `NPer` возвращает значение типа `Double`, оценивающее число периодов ренты, исходя из частоты внесения платежей и предположения о сохранении неизменности платежей и учетной ставки. Она имеет следующий синтаксис:

```
NPer(ставка, платеж, единовременный_платеж[, будущий_платеж[, тип]])
```

где *ставка* — процентная ставка за период; *платеж* — величина выплаты, которая должна производиться каждый период; *единовременный_платеж* — величина единовременно выплачиваемой суммы или текущего значения для серии будущих выплат или поступлений; *будущий_платеж* — необязательный аргумент, равный будущему значению требуемого остатка после последней выплаты; *тип* — необязательный аргумент, задающий время внесения платежей (начало или конец периода).

Первые три аргумента имеют тип `Double`, последние два — `Variant`.

Функция Pmt

Функция `Pmt` возвращает значение типа `Double`, оценивающее величину выплаты ренты, исходя из частоты внесения платежей и предположения о сохранении неизменности платежей и учетной ставки. Она имеет следующий синтаксис:


```
Pmt(ставка, число_периодов, единовременный_платеж[,  
    будущий_платеж[, тип]])
```

где *ставка* — процентная ставка за период; *число_периодов* — общее количество периодов; *единовременный платеж* — величина единовременно выплачиваемой суммы или текущего значения для серии будущих выплат или поступлений; *будущий_платеж* — необязательный аргумент, равный будущему значению требуемого остатка после последней выплаты; *тип* — необязательный аргумент, задающий время внесения платежей (начало или конец периода). Первые три аргумента имеют тип `Double`, последние два — `Variant`.

Функция PPmt

Функция `PPmt` возвращает значение типа `Double`, оценивающее основную выплату ренты за указанный период, исходя из частоты внесения платежей и предположения о сохранении неизменности платежей и учетной ставки. Она имеет следующий синтаксис:

```
PPmt(ставка, период, число_периодов, единовременный_платеж[,  
    будущий_платеж[, тип]])
```

где *ставка* — процентная ставка за период; *период* задает период выплаты в интервале от единицы до *число_периодов* (общего количества периодов); *единовременный платеж* — величина единовременно выплачиваемой суммы; *будущий_платеж* — необязательный аргумент, равный будущему значению требуемого остатка после последней выплаты; *тип* — необязательный аргумент, задающий время внесения платежей (начало или конец периода). Первые четыре аргумента имеют тип `Double`, последние два — `Variant`.

Функция Rate

Функция `Rate` возвращает значение типа `Double`, равное процентной ставке за указанный период. Она имеет следующий синтаксис:

```
Rate(число_периодов, платеж, единовременный_платеж[,  
    будущий_платеж[, тип[, догадка]]])
```

где *число_периодов* — число периодов внесения платежей ренты; *платеж* — величина выплаты, которая должна производиться каждый период; *единовременный_платеж* — величина единовременно выплачиваемой суммы; *будущий_платеж* — необязательный аргумент, равный будущему значению требуемого остатка после последней выплаты; *тип* — необязательный аргумент, задающий время внесения платежей (начало или конец периода); *догадка* — необязательный аргумент, задающий предполагаемое значение функции `Rate` (по умолчанию устанавливается равным 0,1 (10 процентам)).

Функция Syd

Функция `Syd` возвращает значение типа `Double`, обозначающее сумму годовых значений амортизации объекта за определенный период. Она имеет следующий синтаксис:

Syd(стоимость, остаток, время_жизни, период)

где *стоимость* — первоначальная стоимость объекта основных фондов; *остаток* — полезная стоимость объекта по окончании срока полезного использования; *время_жизни* — продолжительность полезного использования объекта; *период* — период амортизации объекта³.

Пример финансовой функции

Порой пользователи забывают о существовании в Access финансовых функций и при необходимости проведения таких расчетов вызывают приложение Excel. При определенных обстоятельствах это решение может быть самым эффективным, однако следует учесть, что и приложение Access имеет те же функции, просто способ хранения исходных данных в Access отличается от Excel.

В качестве примера предположим, что для расширения бизнеса вы решили взять небольшой кредит на приобретение компьютерной техники. Финансовые функции помогут вам рассчитать ежемесячные платежи по любому взятому кредиту. Для того чтобы создать такие функции, вернемся в редактор VBE и откроем модуль Chapter5 (можно открыть новый модуль), после чего:

1. Введите следующую процедуру:

```
Public Function CalPayment(rate As Double, nper As Integer, _
                           pv As Double) As Currency
    CalPayment = Int(Pmt(rate / 12, nper, pv))
End Function
```

Примечание

Обратите внимание на символ подчеркивания в конце первой строки. В языке VBA это — *символ продолжения строки*. Если необходимо продолжить инструкцию на следующей строке, то текущую нужно завершить именно этим символом. Таким образом вы дадите интерпретатору VBA понять, что данную пару строк (или несколько строк) следует расценивать как одну инструкцию и выполнять как единое целое.

2. В окне Immediate введите следующую инструкцию и нажмите клавишу <Enter>.

```
?CalPayment(.07, 48, 10000)
```

На рис. 5.10 показан расчет приблизительных ежемесячных платежей (не учитывающих пошлин), составивший при введенных условиях 240 денежных единиц. Каким же образом VBA определяет, что результат нужно выводить в формате валюты? Посмотрите на первую строку процедуры:

```
Public Function CalPayment(rate As Double, nper As Integer, _
                           pv As Double) As Currency
```

³ Следует особо отметить, что пары параметров *число_периодов* и *ставка*, а также *время_жизни* и *период* в финансовых функциях должны задаваться в одних и тех же единицах измерения (например, месяц или год). — *Примеч. ред.*

Фраза `As Currency` в конце предложения указывает интерпретатору, что данная функция должна возвращать значение денежного типа. Функция `Int` в теле функции обеспечивает целочисленность результата расчетов.

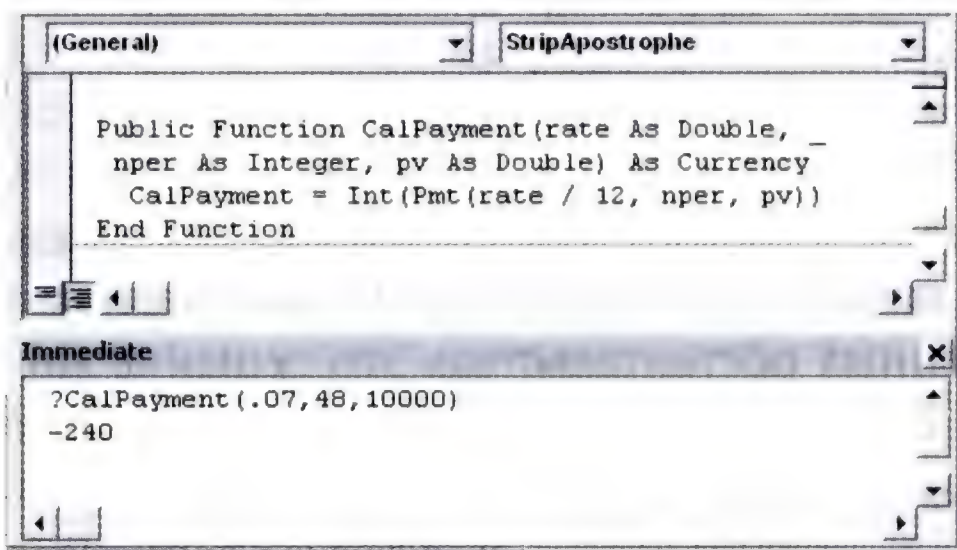


Рис. 5.10. Простая процедура, производящая расчет ежемесячных выплат по ссуде

Использование таких функций поможет вам быстро оценить выгодность покупки товаров в кредит. Для изменения ежемесячных платежей достаточно изменить какие-либо из ее аргументов. Возможно, вы найдете меньшую процентную ставку по кредиту и сможете сэкономить дополнительную сумму денег. Для того чтобы внедрить решение такой задачи в базу данных, потребуется дополнительно создать форму для ввода необходимых значений параметров, пока же мы не использовали никаких элементов интерфейса.

Функции работы со строками

Строковые функции позволяют работать с текстовыми данными. Например, можно заменить одну подстроку другой, узнать, содержит ли строка какой-либо конкретный символ или их комбинацию, и многое другое. Во всех этих случаях вы можете прибегнуть к использованию одной из множества строковых функций, которые будут описаны в настоящем разделе.

Функция Asc

Часто требуется представить строковые символы их целочисленным кодом. Для этого используются кодировки *ANSI* (American National Standards Institute) и *ASCII* (American Standard Code for Information Interchange). В работе с этими десятичными кодами используется функция `Asc`, имеющая следующий синтаксис:

`Asc (строка)`

где *строка* — строковая константа или выражение, состоящие не менее чем из одного символа. Пустые строки (равные `Null`), а также строки нулевой длины (`" "`), переданные в качестве параметра, приводят к ошибке. Возвращаемое значение имеет тип `Integer` и диапазон от 0 до 255.

Примечание

American National Standards Institute (Американский национальный институт стандартизации, или ANSI) — частная неправительственная организация, администрирующая и координирующая добровольную стандартизацию в США.

American Standard Code for Information Interchange (Американский стандарт кодировки для обмена информацией, или ASCII) — это числовое представление символов, которые компьютеры не могут воспринимать как таковые.

Предупреждение

Функция `Asc` возвращает значение типа `Integer` только для первого символа строки независимо от общей ее длины — все остальные символы игнорируются.

Обычно строковое значение должно быть соответствующим образом обрамлено знаками апострофа или двойными кавычками, но этот тезис не относится к функции `Asc`. К примеру, каждая из следующих инструкций вернет значение 49:

```
Asc ("1")
```

```
Asc (1)
```

Совет

Ограничителями называют любые символы, обрамляющие значения. К примеру, в языке VBA принято, что для строковых значений ограничителями являются знаки апострофа и двойных кавычек (например, "Харкинз" или 'Харкинз'), а для значений дат — символы фунта (#).

Функция Chr

Функция `Chr` является обратной функцией к `Asc` и возвращает символ типа `Variant String`, соответствующий числовому коду ASCII, переданному ей в качестве аргумента. Она имеет следующий синтаксис:

```
Chr (код_символа)
```

где *код_символа* — целочисленное значение или выражение типа `Long`. В предыдущем разделе вы узнали, что кодом символа единицы является число 49, так что передача этого числа в качестве параметра в функцию `Chr` даст в результате символ единицы:

```
Chr (49)
```

Совет

Многие строковые функции возвращают значения подтипа `Variant String`. Если вам требуется получить значение типа `String`, дополняйте имя функции символом доллара. К примеру, функция `Chr` вернет значение типа `Variant String`, а функция `Chr$` — типа `String`. В последовательности вычислений более предпочтительным будет последний вариант, так как интерпретатору не придется преобразовывать перед следующей операцией результат этой функции.

Функции изменения регистра

Функции `Lcase` и `Ucase` изменяют регистр символов. Обе эти функции имеют одинаковый синтаксис:

```
Lcase(строка)  
Ucase(строка)
```

где *строка* — строковая константа или выражение, возвращающее строковое значение. Возвращаемое значение имеет тип `Variant String`, однако обе функции имеют варианты со знаком доллара в конце (`Lcase$` и `Ucase$`), возвращающие значения типа `String`. В случае передачи функции в качестве аргумента пустой строки (значение `Null`) функция возвращает значение `Null`. Функция `LCase` преобразовывает все символы строки аргумента в нижний регистр, а функция `UCase` — в верхний.

Функция Len

Функция `Len` используется для подсчета количества символов в строке. Она имеет следующий синтаксис:

```
Len(строка)
```

где *строка* — строковая константа или выражение, возвращающее строковое значение. Результат функции имеет тип `Long Integer`, если аргументом не является `Null`; в противном случае функция возвращает `Null`.

Функции Left, Right и Mid

Три функции — `Left`, `Right` и `Mid` — возвращают из строки аргумента некоторую подстроку, имеющую тип `Variant String` (все три функции имеют версии со знаком доллара, который дополняет их имена, возвращающие значения типа `String`).

Функция `Left` имеет следующий синтаксис:

```
Left(строка, длина)
```

где *строка* — строка, из которой производится извлечение подстроки, а *длина* — количество возвращаемых символов начиная с первого символа строки. Функция `Right` имеет тот же синтаксис, что и `Left`, однако выделяет подстроку, опираясь на конец строки.

Функция `Mid` имеет три аргумента и следующий синтаксис:

```
Mid(строка, начало [, длина])
```

где дополнительный аргумент *начало* определяет символ, с которого выделяется подстрока заданной *длины* (отсчет выполняется слева направо).

Если аргумент *строка* равен значению `Null`, все три функции возвращают `Null`. На рис. 5.11 показан пример, в котором все три функции возвращают разные фрагменты одной и той же строки.

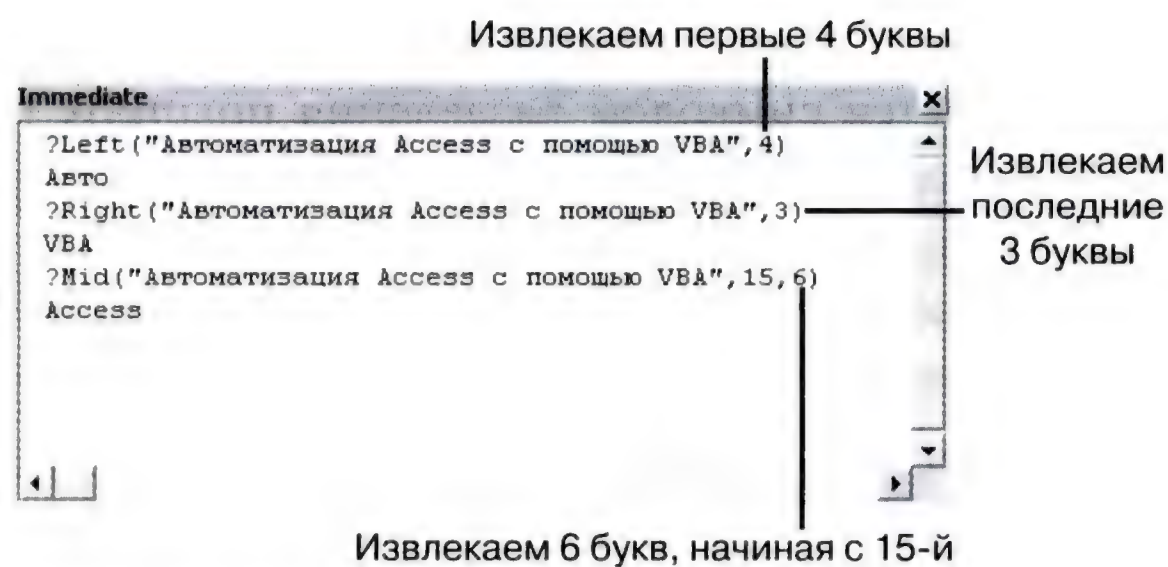


Рис. 5.11. Функции Left, Right и Mid вырезают из строки подстроки

Функция Replace

Функция Replace используется для замены в строке символа или подстроки на другую подстроку. Эта функция имеет следующий синтаксис:

```
Replace(строка, заменяемая_подстрока, заменяющая_подстрока[,  
        начало[, длина[, сравнение]])
```

где строка — исходная строка; заменяемая_подстрока — искомая последовательность символов; заменяющая_подстрока — заменяющая последовательность символов; начало — необязательный аргумент, равный номеру символа, с которого начинается поиск (по умолчанию равен единице); длина — необязательный аргумент, равный количеству выполняемых подстановок (по умолчанию равен -1, что соответствует максимально возможному количеству); сравнение — необязательный аргумент, соответствующий типу сравнения при поиске подстрок (более подробно этот аргумент описан в разделе, посвященном функции Split, далее в этой главе).

Функция Space

Функцию Space используют для формирования строк, состоящих из заданного количества пробелов. Возвращаемое значение имеет тип Variant String. Функция Space имеет следующий синтаксис:

```
Space(длина_строки)
```

где длина_строки — некоторое целочисленное значение, определяющее количество пробелов в формируемой строке. Функция имеет версию с символом доллара (Space\$), возвращающую значение типа String.

Функция Split

Функция Split разбивает строку на массив подстрок, отделенных друг от друга разделителями (количество подстрок должно быть больше единицы). Эта функция имеет следующий синтаксис:

```
Split(выражение[, разделитель[, количество[, сравнение]]])
```


где *выражение* представляет разбиваемую строку; необязательный аргумент *разделитель* является символом разделения подстрок (по умолчанию используется символ пробела); необязательный аргумент *количество* равен числу возвращаемых подстрок (по умолчанию равен -1, что соответствует всем подстрокам); необязательный аргумент *сравнение* указывает на тип сравнения подстрок и может быть равен следующим константам:

- `vbBinaryCompare` — сравнение двоичного представления символов.
- `vbTextCompare` — сравнение по тексту (зависимое от регистра символов).
- `vbDatabaseCompare` — сравнение на основе сведений из базы данных.

Функция StrComp

Функция `StrComp` проверяет сходство двух строк, передаваемых ей в качестве аргументов. Ее синтаксис имеет следующий вид:

```
StrComp(строка1, строка2[, сравнение])
```

где *строка1* и *строка2* — сравниваемые строки, а необязательный аргумент *сравнение* — один из методов сравнения, описанных в предыдущем разделе. Функция, в зависимости от полученных результатов, возвращает следующие целочисленные значения (тип `Variant Integer`):

- -1, если *строка1* < *строки2*;
- 0, если *строка1* = *строке2*;
- 1, если *строка1* > *строки2*;
- Null, если *строка1* или *строка2* имеют значение Null.

Одна строка считается меньше другой, если в результате алфавитного упорядочивания она оказалась в списке перед второй. Функция `StrComp` оказывается полезной в случае, когда нужно отсортировать произвольные данные в алфавитном порядке.

Функции удаления пробелов

Функции `Trim`, `LTrim` и `RTrim` убирают из строки начальные и/или завершающие пробелы. Все они имеют одинаковый синтаксис:

```
Trim(строка)  
RTrim(строка)  
LTrim(строка)
```

где *строка* является строковой константой или выражением. Все три функции возвращают данные типа `Variant String` и поддерживают формат \$, возвращающий результат типа `String`. Если аргумент равен значению Null, все три функции возвращают Null.

Функция `LTrim` удаляет пробелы в начале строки, функция `RTrim` — в ее конце, а `Trim` убирает пробелы с обоих концов строки.

Примечание

Функции фильтрации и агрегирования в настоящей книге не описаны, так как они не являются составной частью объектной модели языка VBA. Эти два типа функций сходны в том, что они основаны на анализе значений полей. Функции фильтрации позволяют определить критерий, ограничивающий число записей, но они не являются родными ни для программы Access, ни для языка VBA. Функции агрегирования выполняют другой тип анализа данных полей, но при этом не ограничивают область рассматриваемых записей. Они являются составной частью языка SQL, но ни в коей мере Access или VBA.

Пример строковой функции

Символы апострофа, часто встречающиеся внутри строк, могут привести к заикливанию программ.

Естественно, апострофы могут попасть в данные по разным причинам, и здесь мы рассмотрим пример функции, их удаляющей. В базе данных примеров TimeTrack зарегистрирована компания “*наб O’Briens*”, имеющая в своем названии символ апострофа. В первую очередь, вернитесь в редактор VBE и откройте модуль Chapter5 (или любой другой стандартный модуль). Теперь выполните такие действия.

1. Введите процедуру:

```
Public Function StripApostrophe(str As String) As String
    StripApostrophe = Replace(str, "'", "")
End Function
```

2. В окне Immediate введите следующую инструкцию и нажмите клавишу <Enter>.

```
?StripApostrophe("наб O’Briens")
```

На рис. 5.12 показан результат выполнения этой функции — как видите, в названии компании апостроф удален. Пользователи базы данных не обязаны знать все тонкости работы с разными типами значений, поэтому разрабатываемые вами средства автоматизации должны очищать вводимые ими данные от проблематичных символов. Естественно, я не предлагаю изменять значения, хранящиеся в самой базе данных, — этими значениями вы можете манипулировать в программах, не изменяя их источник.

Использование функции Format

Форматирование данных обычно отнимает у пользователя немало времени. Осуществить множество predetermined операций форматирования позволяет функция Format. В дополнение к predetermined вы можете создавать собственные способы форматирования и применять их к данным. Потенциальные возможности этой функции сложно охватить в одном разделе, поэтому мы остановимся только на тех, которые всегда необходимо держать в своем арсенале.

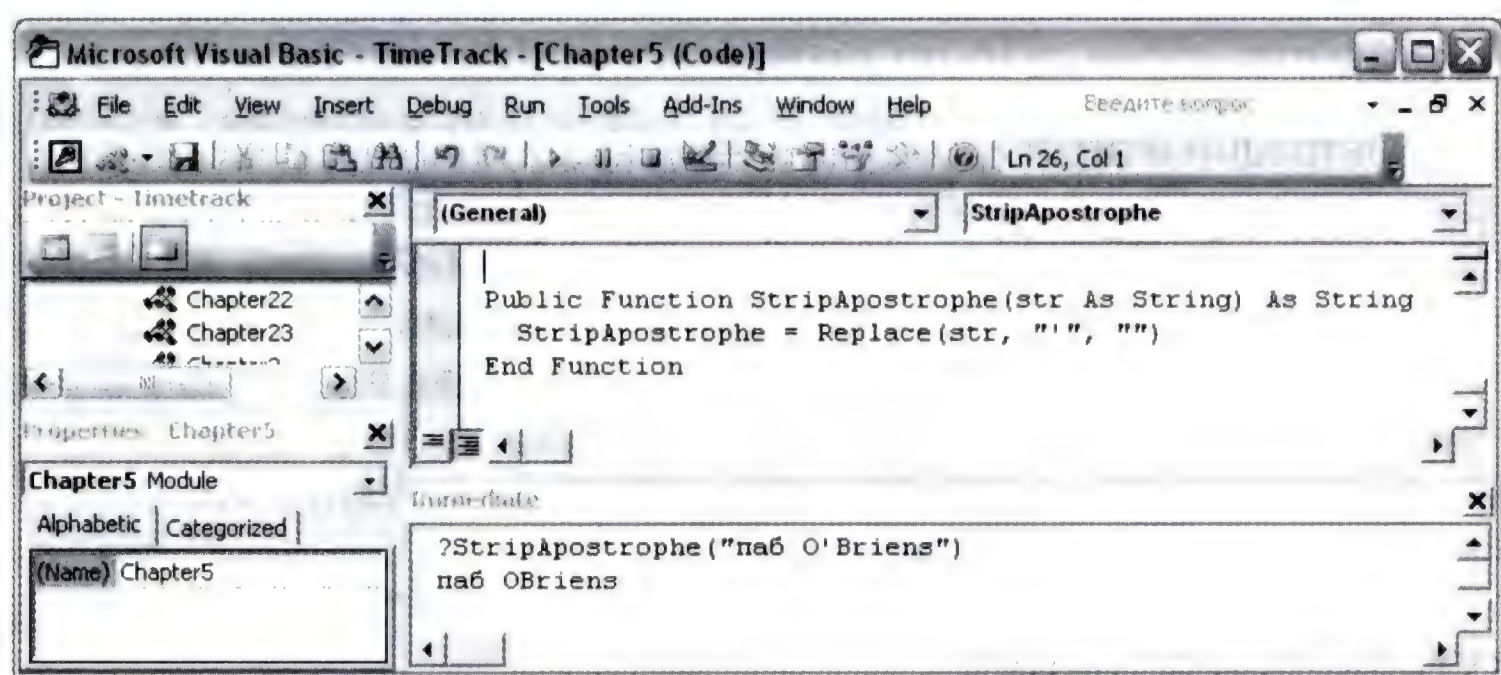


Рис. 5.12. С помощью функции Replace можно удалить проблематичные символы

Предупреждение

Запомните: функция `Format` возвращает значение подтипа `Variant` (т.е. результат и аргумент не имеет всегда один и тот же тип). Это значит, что несмотря на то, что результат работы функции `Format` будет иметь отличный внешний вид, его вряд ли можно будет напрямую использовать с другими данными при вычислениях. Когда это значение вам понадобится для вычислений, попробуйте воспользоваться одной из функций преобразования типов.

Функция `Format` имеет следующий синтаксис:

```
Format (выражение [, формат [, первый_день_недели [,
    первая_неделя_года]]])
```

где *выражение* может иметь строковый или числовой тип и представляет значение, передаваемое в функцию для форматирования. Все остальные аргументы являются необязательными и определяют, как именно *выражение* будет отформатировано. Аргумент *формат* задает один из множества предопределенных и пользовательских способов форматирования. Последние два аргумента были подробно описаны в разделе “Извлечение компонентов даты” ранее в этой главе.

Числа, а также представления времени и даты, серийные номера, строки могут быть отформатированы с помощью функции `Format`; при этом вы можете использовать уникальный набор именованных и определенных пользователем выражений, перечисленных в табл. 5.5 и 5.6.

Многие из выражений форматирования интуитивно понятны, однако некоторые все же требуют дополнительных пояснений. Формат `Currency` основан на региональных настройках в части обозначения валюты, а также символов, используемых в качестве десятичной точки и разделителя групп разрядов. Формат `Fixed` всегда отображает, по крайней мере, одну цифру слева и две справа от десятичной точки. Форматирование `Standard` идентично стилю `Fixed`, за исключением того, что применяет дополнительно разделитель

групп разрядов, определенный в региональных настройках. Формат Percent умножает аргумент на 100 и добавляет справа символ процента.

Таблица 5.5. Именованные числовые форматы

Формат	Пример	Результат
General Number	Format (1234.5678, "General Number")	1234,5678
Currency	Format (1234.5678, "Currency")	1234,56р.
Fixed	Format (0.1, "Fixed")	0,10
Standard	Format (1234.5678, "Standard")	1 234,57
Percent	Format (.5678, "Percent")	56,78%
Scintific	Format (1234.5678, "Scientific")	1,23E+03
Yes/No	Format (0, "Yes/No")	Да
	Format (2, "Yes/No")	Нет
True/False	Format (0, "True/False")	Ложь
	Format (2, "True/False")	Истина
On/Off	Format (0, "On/Off")	Выкл
	Format (2, "On/Off")	Вкл

Таблица 5.6. Именованные форматы даты и времени

Формат	Пример	Результат
General Date	Format ("01/04/04", "General Date")	01.04.2004
Long Date	Format ("01/04/04", "Long Date")	1 Апрель 2004 г.
Medium Date	Format ("01/04/04", "Medium Date")	01-апр-04
Short Date	Format ("01/04/04", "Short Date")	01.04.2004
Long Time	Format ("13:41:02", "Long Time")	13:41:02
Medium Time	Format ("13:41:02", "Medium Time")	01:41
Short Time	Format ("13:41:02", "Short Time")	13:41

Применение форматов, определенных пользователем

Именованные форматы, перечисленные в табл. 5.5 и 5.6, не всегда полностью отвечают потребностям пользователя. Если ни один из них вам не подошел, вы можете создать свой собственный, скомбинировав специальные символы, перечисленные в табл. 5.7–5.9.

Таблица 5.7. Числовые форматы пользователя

Формат	Описание	Пример	Результат
0	Отображения числа или нуля для каждого символа форматирования. Округление, если не хватает места для цифр	Format ("12,3456", "000.00000")	012,34560
		Format ("12,3456", "000.00")	012,35
#	Отображение числа или пустого места. Округление, если не хватает места для цифр	Format ("12,3456", "###.#####")	12,3456
		Format ("12,3456", "###.##")	12,35
%	Умножение числа на 100 и добавление справа знака процента	Format (".3456", "##%")	35%
E- E+ e- e+	Преобразование в научный стиль	Format ("1,234567", "###E-###")	123E-2
		Format ("1,234567", "###e-###")	123e-2
- + \$ ()	Отображение данных символов	Format ("123.45", "\$#####.##")	\$123.45
\	Следующий символ отображается как буква	Format (".3456", "##.##\%")	.35%

Таблица 5.8. Форматы пользователя для дат

Формат	Описание	Пример	Результат
d	Отображение дня месяца без первого нуля	Format ("01/04/04", "d")	1
dd	Отображение дня месяца с добавлением ведущего нуля, если число из первого десятка	Format ("01/04/04", "dd")	01
ddd	Отображение аббревиатуры дня недели	Format ("01/04/04", "ddd")	Чт
dddd	Отображение полного названия дня недели	Format ("01/04/04", "dddd")	четверг

Продолжение табл. 5.8

Формат	Описание	Пример	Результат
dddddd	Отображение короткой даты	Format ("01/04/04", "dddddd")	01.04.2004
ddddddd	Отображение длинной даты	Format ("01/04/04", "ddddddd")	1 Апрель 2004 г.
m	Отображение номера месяца без ведущего нуля	Format ("01/04/04", "m")	4
mm	Отображение номера месяца с добавлением ведущего нуля, если месяц из первого десятка	Format ("01/04/04", "mm")	04
mmm	Отображение аббревиатуры месяца	Format ("01/04/04", "mmm")	апр
mmmm	Отображение полного названия месяца	Format ("01/04/04", "mmmm")	Апрель
q	Отображение номера квартала	Format ("01/04/04", "q")	2
h	Отображение часа без первого нуля	Format ("9:41:02", "h")	9
hh	Отображение часа с добавлением ведущего нуля, если час из первого десятка	Format ("9:41:02", "hh")	09
n	Отображение минут без первого нуля	Format ("9:03:02", "n")	3
nn	Отображение минут с добавлением ведущего нуля, если число минут из первого десятка	Format ("9:03:02", "nn")	03
s	Отображение секунд без первого нуля	Format ("9:03:02", "s")	2
ss	Отображение секунд с добавлением ведущего нуля, если число секунд из первого десятка	Format ("9:03:02", "ss")	02
ttttt	Отображение времени в формате локальных настроек	Format ("13:41:02", "ttttt")	13:41:02

Окончание табл. 5.8

Формат	Описание	Пример	Результат
AM/PM	Отображение времени в 12-часовой системе с аббревиатурами AM или PM (заглавными буквами)	Format ("13:41:02", "hh:mm AM/PM")	01:41 PM
am/pm	Отображение времени в 12-часовой системе с аббревиатурами am или pm (прописными буквами)	Format ("13:41:02", "hh:mm am/pm")	01:41 pm
A/P	Отображение времени в 12-часовой системе с аббревиатурами A или P (заглавными буквами)	Format ("13:41:02", "hh:mm A/P")	01:41 P
a/p	Отображение времени в 12-часовой системе с аббревиатурами a или p (прописными буквами)	Format ("13:41:02", "hh:mm a/p")	01:41 p
ww	Отображение номера недели, от 1 до 54	Format ("01/04/04", "ww")	14
w	Отображение номера дня недели, от 1 до 7	Format ("01/04/04", "w")	5
y	Отображение номера дня в году, от 1 до 366	Format ("01/04/04", "y")	92
yy	Отображение двузначного года, от 00 до 99	Format ("01/04/04", "yy")	04
yyyy	Отображение четырехзначного года, от 0100 до 9999	Format ("01/04/04", "yyyy")	2004

Использование функций семейства Is для беспроблемного выполнения программ

Любая функция может привести к ошибке, если ей передать данные неверного типа. Большинство этих ошибок можно избежать, применив одну из множества встроенных функций VBA семейства Is:

- IsArray. Проверка на массив.
- IsDate. Проверка на тип даты.

- `IsEmpty`. Проверка на инициализацию (т.е. на наличие значения).
- `IsError`. Проверка, не является ли число допустимым значением ошибки.
- `IsMissing`. Проверка, был ли обязательный аргумент передан процедуре.
- `IsNull`. Проверка на пустое значение (равенство `Null`).
- `IsNumeric`. Проверка, является ли результат выражения допустимым числом.
- `IsObject`. Проверка, содержит ли переменная ссылку на объект.

Таблица 5.9. Строковые форматы пользователя

Формат	Описание	Пример	Результат
@	Отображение существующего символа или пробела	<code>Format ("VBA", "@@@@@")</code>	VBA (включая два пробела перед словом)
&	Отображение существующего символа	<code>Format ("VBA", "&&&&")</code>	VBA (без пробелов перед словом)
<	Отображение всех символов в нижнем регистре	<code>Format ("VBA", "<<<")</code>	vba
>	Отображение всех символов в верхнем регистре	<code>Format ("vba", ">>>")</code>	VBA

➔ С массивами вы познакомитесь в соответствующем разделе главы 7.

Все функции семейства `Is` имеют одинаковый синтаксис:

`Is`функция (значение)

где *значение* — это проверяемое значение, переменная, выражение или аргумент. Результатом всех этих функций является булево (логическое) значение. Если условие удовлетворяется, возвращается значение `True`; в противном случае — `False`. К примеру, если значением переменной `varValue` является `Null`, то следующая функция вернет `True`:

`IsNull (varValue)`

Функции взаимодействия с пользователем

Часто в программах требуется ввод какого-то значения пользователем, или отображение ему какой-то информации. В любом из этих случаев приложению придется взаимодействовать с пользователем. Для этих целей чаще всего используются функции `InputBox` и `MsgBox`.

Функция InputBox

В этой главе уже были представлены примеры запроса у пользователя информации, необходимой для выполнения задач. Этот способ взаимодействия с программой достаточно удобен, поскольку невозможно заранее (т.е. при программировании) предугадать все возможные условия задачи; к тому же такое решение позволяет использовать одну и ту же процедуру многократно, подобно параметрическому запросу.

Функция `InputBox` имеет следующий синтаксис:

```
InputBox(приглашение[, заголовок] [, значение_по_умолчанию]  
[, положение_по_x] [, положение_по_y] [, файл_справки, контекст])
```

Единственным обязательным аргументом является *приглашение*. Это строковое значение отображается в диалоговом окне ввода информации и обычно описывает, какую информацию должен ввести пользователь в текстовом поле окна. Максимальная длина строки приглашения составляет 1024 символа. Необязательный аргумент *заголовок* является строкой, отображающейся в заголовке диалогового окна ввода. Если этот аргумент при вызове функции не указан явным образом, в качестве заголовка используется имя приложения. Строковое значение, отображаемое по умолчанию в текстовом поле окна, задается аргументом *значение_по_умолчанию*. Аргументы *положение_по_x* и *положение_по_y* определяют относительное положение диалогового окна на экране (в пикселях относительно верхнего левого угла экрана). Аргументы *файл_справки* и *контекст* используются довольно редко, но обязательно совместно (т.е. либо используются оба сразу, либо не используется ни один). Первый из них указывает на путь к файлу справки, который используется при щелчке пользователем на кнопке справки диалогового окна. Вторым аргументом является число — идентификатором, который назначен каждому разделу справки его автором. (Отметим, что создание файла справки выходит за рамки материала настоящей книги.)

Функция MsgBox

Функция `MsgBox` используется для отображения информации в диалоговом окне. В аргументах мы задаем тип окна и варианты реакции пользователя. Эта функция возвращает целочисленное значение и имеет следующий синтаксис:

```
MsgBox(приглашение[, кнопки] [, заголовок] [, файл_справки,  
контекст])
```

Аргумент *приглашение* является обязательным. Он имеет строковый тип данных и содержит сообщение, отображаемое в диалоговом окне. Состав командных кнопок окна определяется необязательным аргументом *кнопки* (в табл. 5.10 перечислены все возможные константы, определяющие этот параметр). Если вы хотите задать для окна собственный заголовок, передайте его необязательному аргументу *заголовок*. Аргументы *файл_справки* и *контекст* имеют то же предназначение и спецификацию, что и в функции `InputBox` (см. предыдущий раздел).

Таблица 5.10. Константы состава кнопок функции MsgBox

Константа	Описание кнопки	Целочисленное значение
vbOKOnly	Только кнопка ОК	0
vbOKCancel	Кнопки ОК и Отмена	1
vbAbortRetryIgnore	Кнопки Прервать, Повтор и Пропустить	2
vbYesNoCancel	Кнопки Да, Нет и Отмена	3
vbYesNo	Кнопки Да и Нет	4
vbRetryCancel	Кнопки Повтор и Отмена	5

В табл. 5.11 перечислены значки, которые могут быть отображены в диалоговом окне, и соответствующие им константы. Аргумент *кнопки* формируется как сумма констант кнопок и значков, т.е. как *константа_кнопки + константа_значка*

Например, вызов функции
MsgBox "Отмеченные записи будут удалены. Продолжить?", vbOKCancel + _
vbCritical, "Предупреждение!"
приведет к открытию диалогового окна, показанного на рис. 5.13. В нем содержатся две кнопки: ОК и Отмена, а также значок предупреждения в виде белого креста в красном круге. Суммирование констант vbOKCancel и vbCritical позволило отобразить в диалоговом окне кроме текста сообщения еще и несколько элементов.

Таблица 5.11. Константы значков

Константа	Значок	Целочисленное значение
vbCritical	Белый крестик в красном круге	16
vbQuestion	Вопросительный знак в голубом пузырьке	32
vbExclamation	Восклицательный знак в желтом треугольнике	48
vbInformation	Буква i в голубом пузырьке	64

Возвращаемое функцией значение отражает действие пользователя, произведенное в окне, и зависит от конкретной нажатой кнопки. В табл. 5.12 перечислены все возможные значения кнопок.

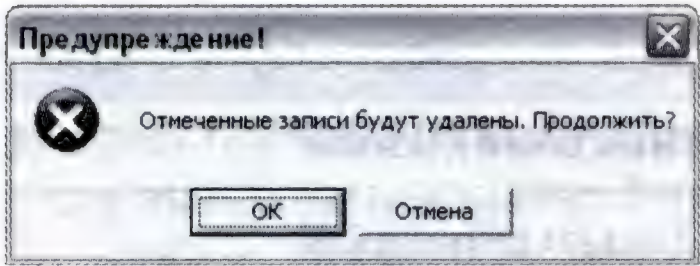


Рис. 5.13. При вызове окна определяется состав кнопок и отображаемый в нем значок

Таблица 5.12. Значения кнопок

Нажатая кнопка	Возвращаемое значение	Целочисленное значение
ОК	vbOK	1
Отмена (или Esc)	vbCancel	2
Прервать	vbAbort	3
Повтор	vbRetry	4
Пропустить	vbIgnore	5
Да	vbYes	6
Нет	vbNo	7

В примере, показанном на рис. 5.13, окно сообщения имеет две кнопки — ОК и Отмена. Это значит, что функция может вернуть одно из двух значений: 1 или 2.

ПРАКТИКУМ

Подсчет рабочих дней

Частью управления проектами в базе данных TimeTrack.mdb является оценка количества рабочих дней, которые отведены на выполнение некоторой задачи. Встроенная функция VBA DateDiff быстро подсчитает общее количество календарных дней, но не существует такой функции, которая вычла бы из этого числа выходные.

В предлагаемом решении мы добавляем в форму Projects командную кнопку и текстовое поле. При щелчке на этой кнопке в новом текстовом поле отображается количество рабочих дней между датами начала и планового завершения проекта. Это позволит пользователю более точно оценить потенциально возможные трудозатраты.

- 1. Откройте форму Projects в режиме конструктора и добавьте в нее командную кнопку и текстовое поле справа от элемента Дата окончания. Присвойте командной кнопке имя cmdCalculate, а текстовому полю — txtBusiness-Days. Установите значение свойства Вывод на экран текстового поля в Нет.
- 2. Щелкните на кнопке Code (программа) панели инструментов. Откроется окно редактора VBE и в нем окно модуля формы.

3. Введите следующую процедуру обработки события:

```

Private Sub cmdCalculate_Click()
'Определяет количество рабочих дней,
'отведенных на выполнение проекта,
'исключая из общего количества выходные
Dim dtmStart As Date
Dim dtmEnd As Date
Dim intTotalDays As Integer
Dim intWeekendDays As Integer
Dim intBusinessDays As Integer
If IsNull(StartDate) Then
    MsgBox "Введите дату начала проекта", _
        vbOKOnly, "Ошибка"
    Exit Sub
End If
If IsNull(EstimatedEndDate) Then
    MsgBox "Введите дату окончания проекта", _
        vbOKOnly, "Error"
    Exit Sub
End If
dtmStart = StartDate
dtmEnd = EstimatedEndDate
Select Case DatePart("w", dtmStart, vbMonday)
    Case Is = 6
        dtmStart = DateAdd("d", dtmStart, 2)
    Case Is = 7
        dtmStart = DateAdd("d", dtmStart, 1)
End Select
Select Case DatePart("w", dtmEnd, vbMonday)
    Case Is = 6
        dtmEnd = DateAdd("d", dtmEnd, -1)
    Case Is = 7
        dtmEnd = DateAdd("d", dtmEnd, -2)
End Select
intTotalDays = DateDiff("d", dtmStart, dtmEnd) + 1
intWeekendDays = DateDiff("ww", dtmStart, dtmEnd, vbMonday) * 2
intBusinessDays = intTotalDays - intWeekendDays
txtBusinessDays = intBusinessDays
txtBusinessDays.Visible = True
End Sub

```

4. Сохраните программу и вернитесь в Access. Для этого щелкните на кнопке **View** **Microsoft Office Access** панели инструментов редактора VBE или на кнопке формы **Projects** панели задач операционной системы.

5. Щелкните на кнопке **Вид** панели инструментов Access, и вы увидите форму, показанную на рис. 5.14.

6. Теперь щелкните на кнопке **Исключать выходные**. Справа от кнопки отобразится текстовое поле, в котором будет выведено количество рабочих дней от начала до конца проекта (рис. 5.15). Не волнуйтесь, если в вашей форме будет вычислено отличное от рисунка значение, поскольку фактическое количество дней зависит от конкретного дня (а не от дня, в который выполнялись вычисления).

The 'Projects' form contains the following fields and controls:

- № проекта: 4
- Клиент: Сенатор-Авто
- Название проекта: Управление списком рассылк
- Дата начала: 24.02.2004
- Дата окончания: 29.12.2004
- Исключать выходные: (checkbox)

	TaskID	TaskName	HourlyRate
+	25	Управление проектом	150,00р.
+	26	Разработка	105,00р.
+	27	Тестирование	65,00р.
+	28	Сопровождение	45,00р.
*	(Счетчик)		

Navigation controls at the bottom include record navigation buttons and a status bar showing 'Запись: 1 из 4' and 'Запись: 3 из 30'.

Рис. 5.14. Новое текстовое поле на этой форме не показано, поскольку его свойство Вывод на экран было установлено в значение Нет

This screenshot shows the same 'Projects' form as Figure 5.14, but with the 'Исключать выходные' button highlighted. To the right of the button, the value '222' is displayed, representing the calculated number of working days between the start and end dates.

Рис. 5.15. Щелкните на новой командной кнопке для вычисления и отображения количества рабочих дней между датами начала и конца проекта

После объявления и инициализации множества переменных с помощью функции IsNull мы проверяем наличие введенных дат начала и конца проекта. Если какое-либо из этих значений отсутствует, открывается окно предупреждения и производится выход из процедуры.

Если обе даты пользователем введены, первая инструкция Select Case корректирует дату начала проекта, если та попадает на субботу или воскресенье, добавляя к ней, соответственно, двойку или единицу. Вторая инструкция Select Case корректирует дату завершения проекта, и в случае попадания ее на выходные вычитает единицу или двойку.

- ➔ Конструкции `Select Case` и `If End` будут описаны в соответствующих разделах главы 6.
- ➔ В описанном примере процедура вызывается при возникновении события в элементе управления. Подробно процесс обработки событий описывается в главе 11.

Все оставшиеся строки программы посвящены подсчету общего количества дней и количества выходных (суббот и воскресений) между двумя скорректированными датами. Количество рабочих дней является результатом вычитания из первого значения второго.

В процедуре не установлена явная проверка того, является ли дата завершения проекта больше даты его начала. Вы сами должны решить, хотите ли вы ограничивать дату завершения, и в случае необходимости установить на соответствующее поле допустимый диапазон значений.

Совет

В главе 3 говорилось о важности установки корректного типа данных для каждого поля. Теперь вы уже знаете, по крайней мере, одно из достоинств этого правила. Не беспокойтесь о возможности некорректного ввода дат в текстовые поля, поскольку в них установлен тип данных `Date/Time`. При попытке ввода в текстовое поле информации, отличной от даты, будет выдано сообщение об ошибке “Введенное значение не подходит для данного поля”. Эту задачу обработки ошибки мы пропускаем, поскольку само приложение на уровне таблицы исключает ее возникновение.

Какова же роль процедуры обработки события `Current`? Она очищает содержимое текстового поля и устанавливает его свойство отображения на экране в `Нет` при перемещении к другой записи таблицы проектов.

В данной процедуре использовано множество хороших приемов, о которых говорилось в главе 2. В частности, каждая строка имеет соответствующий отступ, проясняющий структуру программы, а в начале процедуры вставлен комментарий, описывающий ее назначение. В дополнение, в процедуре явным образом объявляются переменные разных типов.

Использование инструкций передачи управления

6

В ЭТОЙ ГЛАВЕ

Циклы и разветвления	119
Конструкция If...Then...Else	120
Конструкция Select Case.....	122
Конструкция цикла For...Next	123
Конструкция цикла Do	128

Циклы и разветвления

До сих пор практически все программы, которые вы встречали в этой книге, были строго последовательными. Это значит, что выполнение инструкций начиналось в начале процедуры и проходило построчно до ее конца. Единственным отступлением от этого правила была обработка ошибок.

➔ Процедуры обработки ошибок рассматривались в одноименном разделе главы 4.

В то же время язык VBA позволяет реализовывать и более сложные модели выполнения программ. Для того чтобы уметь писать гибкие программы на этом языке, нужно понять концепции *разветвления* и *цикла*. Разветвление позволяет интерпретатору VBA принимать решения и выполнять в зависимости от них определенную последовательность инструкций. Циклы же позволяют выполнять некоторую последовательность инструкций множество раз. Существует несколько вариаций каждого из этих понятий. Совместно же обе эти структуры (разветвление и циклы) известны как *инструкции передачи управления*, поскольку они определяют порядок выполнения инструкций в программе.

Конструкция If...Then...Else

Первой инструкцией передачи управления является оператор If (“если”), который позволяет выполнять определенный фрагмент программы только в том случае, если справедливо некоторое заданное условие.

Простая инструкция If

Рассмотрим использование оператора If для управления выполнением одной или нескольких инструкций, например:

```
Function IsSunday(dtmDayToCheck As Date) As Boolean
    ' Возвращает значение True, если день недели -- воскресенье
    IsSunday = False
    If DatePart("w", dtmDayToCheck) = 1 Then
        IsSunday = True
    End If
End Function
```

Этой функции передается единственный аргумент типа даты (Date), возвращаемое же ею значение имеет логический тип (Boolean). Если функция DatePart в запросе на день недели возвращает единицу (соответствующую воскресенью), то функция возвращает значение True; в противном случае — False (это значение присваивается процедуре по умолчанию в ее первой строке).

Схематически простейшая форма оператора If имеет следующую конструкцию:

```
If условие Then
    инструкции
End If
```

Условием может быть любое выражение, возвращающее значение True или False. Если условие истинно (True), то последовательно выполняются инструкции, находящиеся до оператора End If; в противном случае все эти строки опускаются, а управление передается первой инструкции, следующей за оператором End If.

Совет

Если *условие* возвращает числовое значение, значение 0 расценивается как False, любое другое — как True.

Создание более сложных условий

Условия могут использовать логику любой сложности с применением при необходимости круглых скобок группировки. В качестве примера рассмотрим функцию, возвращающую истинное значение, если аргумент даты приходится на выходной день недели (субботу или воскресенье).

```
Function IsWeekend(dtmDayToCheck As Date) As Boolean
    ' Возвращает True, если данное число является выходным днем недели
    IsWeekend = False
    If ((DatePart("w", dtmDayToCheck) = 1) Or _
```



```
        (DatePart("W", dtmDayToCheck) = 7)) Then
        IsWeekend = True
    End If
End Function
```

В приведенной процедуре условие разбито на две строки. Если функция `DatePart` возвращает значение `True` хотя бы в одном случае, функция возвращает значение `True`; в противном случае значение `False`, установленное по умолчанию, не изменяется.

Чаще всего в логических выражениях можно встретить следующие ключевые слова:

- **Or.** Это ключевое слово используется, когда требуется выполнение хотя бы одного из пары условий. Если в логическом выражении участвует больше двух условий, и все они объединены оператором `Or`, все выражение принимает значение `True`, если истинно хотя бы одно из условий.
- **And.** Это ключевое слово используется, когда требуется выполнение всех условий в выражении. Если этим оператором объединены два условия, то значение `True` достигается только в том случае, если выполнены оба условия.
- **Not.** Этот оператор преобразует значение `True` в `False` и наоборот.

В табл. 6.1 показаны примеры использования логических ключевых слов.

Таблица 6.1. Примеры использования логических ключевых слов

Условие	Разъяснение
(intDays = 4) Or (intWeeks = 3)	Истинно, если intDays равно четырем, или intWeeks равно трем, или выполняются оба условия одновременно
(intDays = 4) And (intWeeks = 3)	Истинно, если одновременно intDays равно четырем и intWeeks равно трем
Not (intDays = 4)	Истинно, если intDays имеет значение, отличное от четырех

Добавление инструкции Else

Существует несколько необязательных элементов конструкции `If...End If`. Первым из них является инструкция `Else`.

```
Function IsWeekday(dtmDayToCheck As Date) As Boolean
    ' Возвращает значение True, если аргумент является
    ' рабочим днем недели
    If ((DatePart("w", dtmDayToCheck) = 1) Or _
        (DatePart("W", dtmDayToCheck) = 7)) Then
        IsWeekday = False
    Else
        IsWeekday = True
    End If
End Function
```



```
End If  
End Function
```

Добавление инструкции `Else` позволило вернуть другое значение при невыполнении условия. Схематически общая конструкция оператора `If` стала следующей:

```
If условие Then  
    инструкции1  
Else  
    инструкции2  
End If
```

Если условие истинно, выполняется первый набор инструкций, если ложно — то второй.

Использование инструкции `ElseIf`

Еще одной необязательной частью структуры `If...End If` является инструкция `ElseIf`. Вот пример ее использования:

```
Function GetDayName(dtmDayToCheck) As String  
    ' Возвращает день недели, соответствующий переданной дате  
    If DatePart("w", dtmDayToCheck) = 1 Then  
        GetDayName = "Воскресенье"  
    ElseIf DatePart("w", dtmDayToCheck) = 2 Then  
        GetDayName = "Понедельник"  
    ElseIf DatePart("w", dtmDayToCheck) = 3 Then  
        GetDayName = "Вторник"  
    ElseIf DatePart("w", dtmDayToCheck) = 4 Then  
        GetDayName = "Среда"  
    ElseIf DatePart("w", dtmDayToCheck) = 5 Then  
        GetDayName = "Четверг"  
    ElseIf DatePart("w", dtmDayToCheck) = 6 Then  
        GetDayName = "Пятница"  
    Else  
        GetDayName = "Суббота"  
    End If  
End Function
```

Удостоверьтесь, что в эту функцию вы передаете переменную типа даты, или константу, ограниченную символами `#`, иначе функция независимо от аргумента вернет значение `Суббота`. При обработке интерпретатором этой конструкции все условия проверяются последовательно, начиная со следующего за оператором `If`, и продолжая всеми, следующими за `ElseIf`. Если некоторое условие оказывается истинным, выполняется соответствующая ему группа инструкций, после чего управление передается оператору `End If`. Основным отличием операторов `Else` и `ElseIf` является возможность использования нескольких условий вместо одного.

Конструкция `Select Case`

Если в конструкции `If` сложно разобраться ввиду наличия множества элементов `ElseIf`, для упрощения используют практически такую же конструк-

цию — `Select Case`. С ее использованием приведенный выше пример можно переписать в следующем виде:

```
Function GetDayName(dtmDayToCheck) As String
    ' Возвращает день недели, соответствующий переданной дате
    Select Case DatePart("w", dtmDayToCheck)
        Case 1
            GetDayName = "Воскресенье"
        Case 2
            GetDayName = "Понедельник"
        Case 3
            GetDayName = "Вторник"
        Case 4
            GetDayName = "Среда"
        Case 5
            GetDayName = "Четверг"
        Case 6
            GetDayName = "Пятница"
        Case Else
            GetDayName = "Суббота"
    End Select
End Function
```

Для использования структуры `Select Case` в саму инструкцию `Select` подставляется некоторое выражение, возвращающее значение. За этой инструкцией следует несколько инструкций `Case`, каждая из которых содержит некоторое значение. Интерпретатор сравнивает результат выражения `Select` со всеми значениями в инструкциях `Case`, и в случае их равенства выполняет набор инструкций, следующий за данным оператором вплоть до следующего `Case` или `End Select`. Следует заметить, что можно использовать оператор `Case Else`, соответствующий всем тем значениям, которые явно не указаны ни в одном операторе `Case` текущей конструкции.

Совет

Хорошей практикой является использование в конструкциях `Select` оператора `Case Else`, поскольку это позволяет избежать несоответствия заданного значения ни одному из проверяемых. Такой подход позволяет избежать многих потенциальных ошибок данных и ошибок выполнения программы.

Конструкция цикла For...Next

Иногда необходимо многократное выполнение некоторого фрагмента программы. Именно для этой цели в языке VBA предусмотрены операторы цикла. Первой конструкцией цикла, которую мы рассмотрим, будет `For . . . Next`.

Для того чтобы увидеть все удобства, которые приносят циклические операторы, рассмотрим следующий пример, в котором циклов еще нет:

```
Sub PrintWeek(dtmStart As Date)
    ' Выводит неделю, начинающуюся с даты
    ' переданной в аргументе dtmStart
```



```

Debug.Print dtmStart
Debug.Print DateAdd("d", 1, dtmStart)
Debug.Print DateAdd("d", 2, dtmStart)
Debug.Print DateAdd("d", 3, dtmStart)
Debug.Print DateAdd("d", 4, dtmStart)
Debug.Print DateAdd("d", 5, dtmStart)
Debug.Print DateAdd("d", 6, dtmStart)
End Sub

```

На рис. 6.1 показан результат выполнения этой процедуры. В окне Immediate распечатана последовательность из семи дат, начинающаяся с заданной. В реальной жизни результаты подобных расчетов записываются в файл или выводятся на принтер, однако в нашем распоряжении пока еще нет соответствующих инструментов языка VBA.

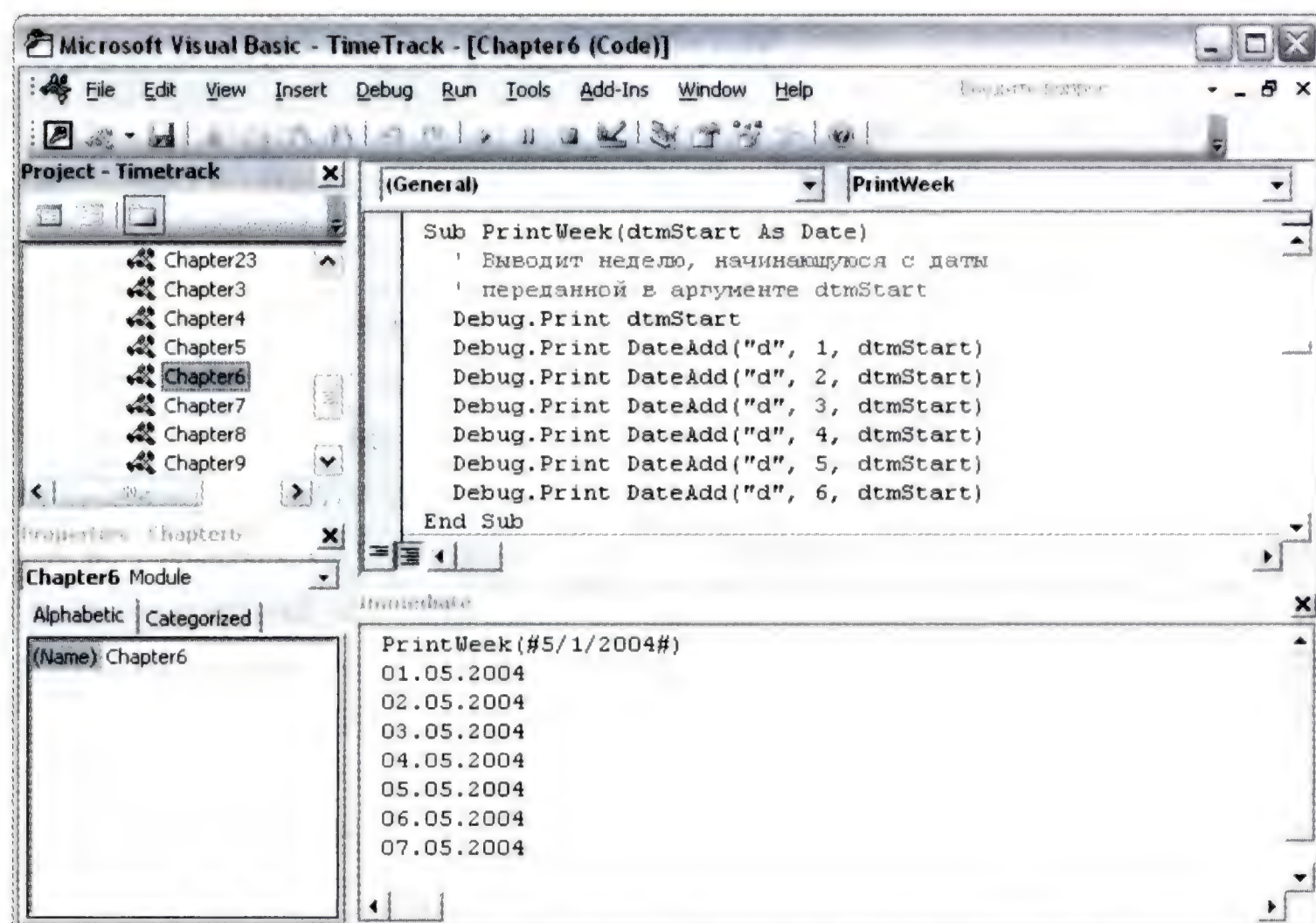


Рис. 6.1. Вывод последовательности дат без использования цикла

Несмотря на то, что данная процедура выполняет то, для чего она предназначалась, в ней налицо две явные проблемы. Во-первых, операторы многократно повторяются; в результате текст программы даже такой простой задачи становится слишком длинным. Во-вторых, данная процедура не гибка: что делать, если, к примеру, потребуется распечатать не 7, а тридцать последующих чисел? При этом, естественно, придется менять текст всей программы.

Обе эти проблемы можно обойти путем использования структуры For...Next. Вот та же процедура, но описанная уже с помощью цикла:

```

Sub PrintWeek(dtmStart As Date)
    ' Выводит неделю, начинающуюся с даты
    ' переданной в аргументе dtmStart
    Dim intI As Integer
    For intI = 0 To 6
        Debug.Print DateAdd("d", intI, dtmStart)
    Next intI
End Sub

```



```
Next intI
End Sub
```

В данной процедуре переменную `intI` называют *счетчиком цикла*. В структуре `For...Next` ей присваиваются значения от 0 до 6, и выполняются инструкции, заключенные между операторами `For` и `Next`, для каждого из этих значений. В данном примере телом цикла является всего один оператор, однако их потенциальное количество не ограничено.

Примечание

Как правило, счетчикам цикла присваивают простые короткие имена типа `i`, `j` или `k`. Если вы используете некоторое соглашение об именах, можете аналогично использовать `intI`, `intJ` или `intK`. В то же время некоторые программисты и для счетчиков используют значимые имена типа `intDays`.

Совет

В операторе `Next` не обязательно упоминать имя переменной счетчика цикла. Как правило, это делается исключительно из соображения понятности текста программы. В то же время не удивляйтесь, если в некоторых программах вы не встретите имени переменной в этом операторе.

Циклы с обратным отсчетом

Цикл, который мы только что создали, работает с последовательностью чисел 0, 1, 2, 3, 4, 5 и 6, являющимися значениями счетчика. По умолчанию конструкция `For...Next` прибавляет в каждом цикле к переменной счетчика единицу; в то же время при желании вы можете этот порядок изменить, установив собственное значение приращения. Вот пример отсчета значений в обратной последовательности:

```
Sub PrintWeekReverse(dtmStart As Date)
    ' Выводит неделю, заканчивающуюся датой
    ' переданной в аргументе dtmStart
    Dim intI As Integer
    For intI = 6 To 0 Step -1
        Debug.Print DateAdd("d", intI, dtmStart)
    Next intI
End Sub
```

В данном случае часть `Step` оператора `For` указывает интерпретатору VBA, что из значения счетчика цикла нужно каждый раз вычитать единицу; однако вы можете изменить величину приращения на любую другую. В следующем примере значение счетчика в каждом цикле будет увеличиваться на 3:

```
For intI = 0 to 9 Step 3
    инструкции
Next intI
```

В данном случае *инструкции* в цикле будут выполняться четыре раза — при значениях счетчика 0, 3, 6 и 9.

Использование для счетчика цикла переменных границ

Во второй версии процедуры PrintWeek счетчик цикла проходит фиксированный набор значений от нуля до шести, однако эти границы могут быть и переменными. Рассмотрим более гибкую процедуру:

```
Sub PrintDays(dtmStart As Date, intDays As Integer)
    ' Выводит несколько дней, начинающихся датой,
    ' переданной в аргументе dtmStart
    Dim intI As Integer
    For intI = 0 To intDays - 1
        Debug.Print DateAdd("d", intI, dtmStart)
    Next intI
End Sub
```

Теперь количество выводимых на печать дат определяется вторым аргументом процедуры. На рис. 6.2 показан пример вызова этой процедуры для печати нескольких последовательных дат.

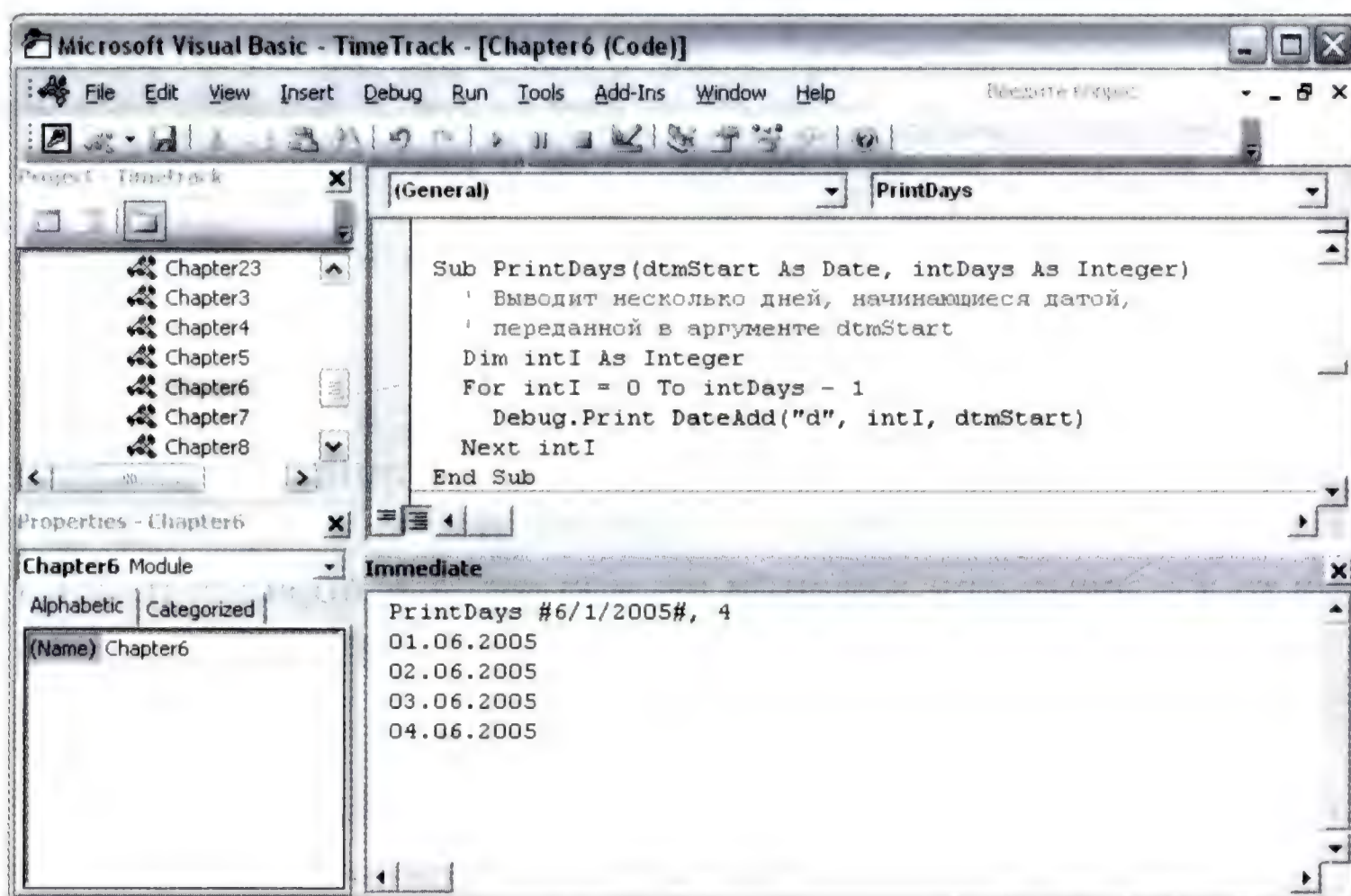


Рис. 6.2. Печать последовательности дат в цикле

Вложенные циклы For...Next

Одни циклы могут в своем теле содержать другие. Вложенные циклы позволяют дополнительно сократить объем программы. Предположим, что нам нужно вывести на печать карточки времени для каждого рабочего часа недели. Это можно реализовать путем вложения цикла по рабочим часам в цикл по рабочим дням:

```
Sub PrintTimecard(dtmStart As Date)
    ' Распечатка карточек времени для рабочих часов недели
    Dim intI As Integer
    Dim intJ As Integer
    For intI = 0 To 6
        Debug.Print DateAdd("d", intI, dtmStart)
```



```

For intJ = 0 To 8
    Debug.Print " " & DateAdd("h", intJ, #9:00:00 AM#)
Next intJ
Next intI
End Sub

```

Рассмотрим вкратце работу этой программы. Внешний цикл со счетчиком `intI` проходит по всем семи дням недели. Для каждого значения `intI` выводится дата, после чего управление передается внутреннему циклу. Внутренний цикл выполняется девять раз, после чего управление возвращается внешнему циклу. Если с визуализацией результата у вас возникли проблемы, посмотрите на рис. 6.3, где показан результат работы процедуры.

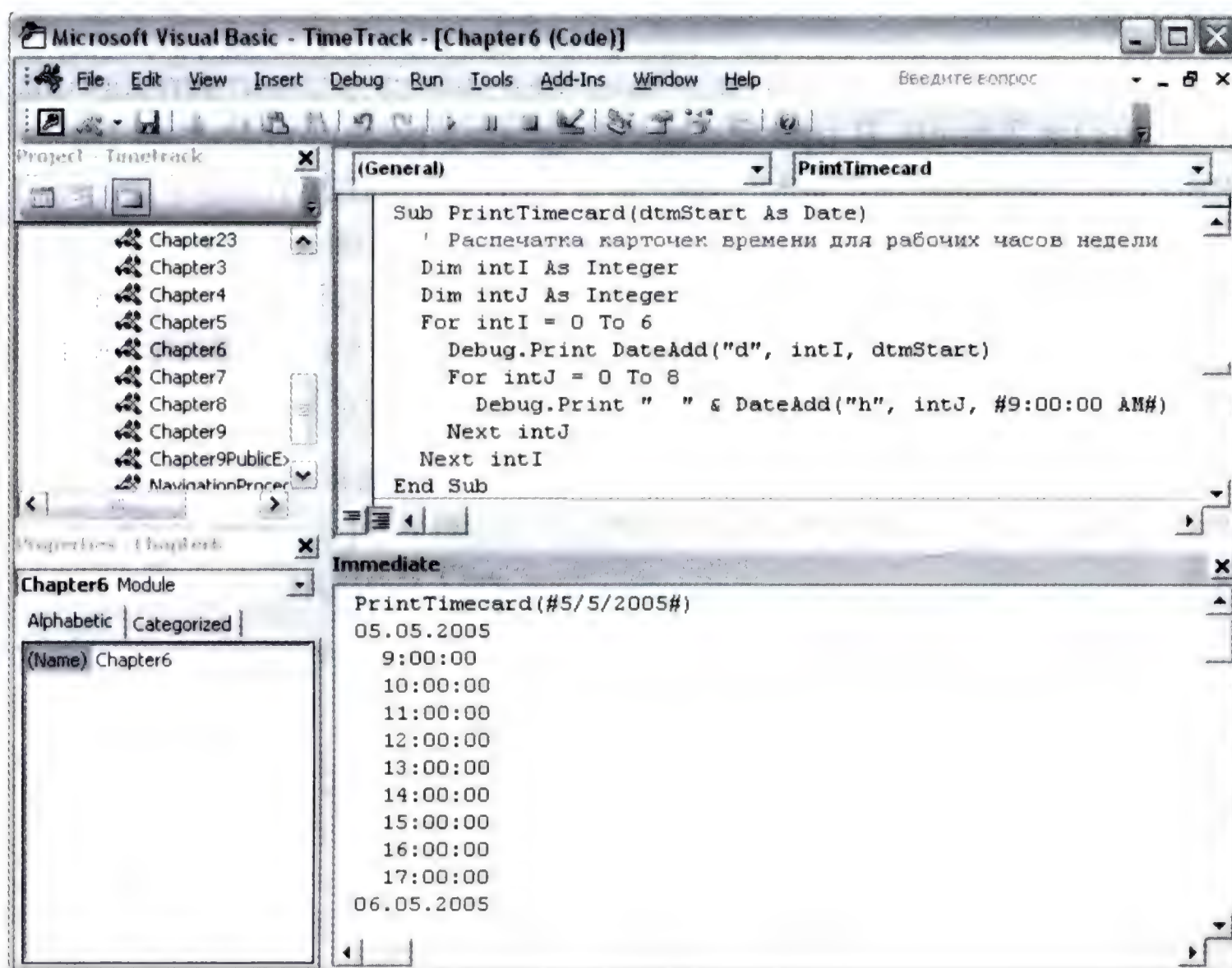


Рис. 6.3. Использование вложенных циклов

Эта процедура выводит на печать 70 строк. Используя вложенные циклы, мы достигаем цели с помощью всего 8 строк программы вместо 70 вызовов функции `Debug.Print`.

Совет

Программист не ограничен возможностью вложения всего двух циклов — если с точки зрения процедуры это целесообразно, он может вложить друг в друга три, четыре, пять и более циклов.

Выход из цикла For...Next

Иногда в ходе выполнения цикла возникают ситуации, когда нужно из него выйти преждевременно, не достигая максимального значения счетчика.

Для этих целей в языке VBA предусмотрен оператор `Exit For`. Следующая процедура возвращает дату понедельника, ближайшего к заданному числу.

```
Function GetNextMonday(dtmDateStart As Date) As Date
    ' Возвращает дату следующего понедельника
    Dim intI As Integer
    Dim dtmDayToCheck As Date
    For intI = 1 To 7
        dtmDayToCheck = DateAdd("d", intI, dtmDateStart)
        If DatePart("w", dtmDayToCheck) = 2 Then
            GetNextMonday = dtmDayToCheck
            Exit For
        End If
    Next intI
End Function
```

Совершенно очевидно, что понедельник не может отстоять от любой заданной даты более чем на 7 дней. В приведенной процедуре используется цикл от 1 до 7, постепенно увеличивающий заданную дату с помощью функции `DateAdd`. В каждом цикле производится проверка, не является ли новая дата понедельником, и если это так, то выполняется выход из цикла `For...Next`. Исполнение инструкции `Exit For` приводит к передаче управления к следующей за циклом инструкции (в данном примере это оператор `End Function`).

➔ Еще одна форма цикла — конструкция `For...Each` — работает с переменными объектов. Подробная информация о ней изложена в разделе “Работа с наборами” главы 8.

Конструкция цикла Do

Циклы `For...Next` оказываются полезными в тех случаях, когда заранее известно число повторений выполнения некоторой последовательности инструкций. Однако зачастую это число предварительно не известно, и именно тогда на помощь приходит конструкция цикла `Do`.

Простой цикл Do

В следующем примере выводится таблица, показывающая сумму счета, которая должна выставляться за один, два и больше (вплоть до восьми) часов работы консультанта.

```
Sub PrintBilling(curHourlyRate As Currency)
    ' Вывод таблицы стоимости работ
    Dim intHours As Integer
    intHours = 1
    Do
        Debug.Print intHours * curHourlyRate
        intHours = intHours + 1
    Loop Until intHours = 9
End Sub
```

Строки между операторами `Do` и `Loop` содержат выполняемые в цикле инструкции. Когда интерпретатор VBA встречает оператор `Do`, он сразу перехо-

дит к строке следующей за ним инструкции и выполняет тело цикла до тех пор, пока условие в операторе Loop не примет значение True. В данном примере выполнение цикла повторяется до тех пор, пока переменная `intHours` (Количество часов) не станет равной 9, после чего управление передается следующей за Loop инструкцией.

Совет

Если в программировании цикла `Do...Loop` допущена ошибка (например, если вы забыли в теле цикла изменять параметр, по которому проверяется условие выхода из цикла), может получиться бесконечный цикл, который будет выполняться постоянно, практически блокируя работу компьютера. Если вы столкнулись с такой ситуацией, нажмите клавиши `<Ctrl+Break>`, чтобы прервать выполнение процедуры или перейти в режим отладки.

Примечание

Во многих случаях конструкцию `Do...Loop` можно заменить конструкцией `For...Next`. В примере функции `PrintBilling` в качестве структуры цикла можно было использовать инструкцию `For intHours = 1 To 8`. При выборе структуры цикла прежде всего думайте о том, какая из них будет более понятной (и логичной) в тексте программы.

Разновидности циклов Do

Циклические конструкции Do имеют несколько немного отличающихся разновидностей. Можно установить проверку условия как в начале, так и в конце цикла и выполнять его пока выполняется, а также пока *не* выполняется заданное условие. В первом примере данного цикла условие проверялось в конце цикла, а сам цикл продолжал выполняться до тех пор, пока условие не выполнялось. В общем случае такая конструкция имеет следующий вид:

```
Do
    инструкции
Loop Until условие
```

При желании вы можете выполнять инструкции цикла до тех пор, пока условие продолжает выполняться. В следующем примере в конце каждого цикла проверяется выполнение условия, и если результат проверки становится отрицательным, происходит выход из цикла.

```
Do
    инструкции
Loop While условие
```

Аналогично, можно проверять условие и в начале цикла. Это можно сделать как с помощью условия `Until`:

```
Do Until условие
    инструкции
Loop
```


так и помощью условия While:

```
Do While условие
    инструкции
Loop
```

Выбор между этими четырьмя вариантами конструкции Do . . . Loop может оказаться весьма проблематичным. Если оказалось, что цикл выполняется не то количество раз, которое вы изначально запланировали, проверьте, ту ли форму конструкции Do . . . Loop вы применили. Здесь могут понадобиться средства отладки, о которых говорилось в главе 4. Пошагово проходя по инструкциям цикла, вы реально сможете понять, что на самом деле происходит.

Предупреждение

Если проверка условия вставлена в конце цикла, этот цикл будет выполнен хотя бы один раз независимо от начального значения условия.

Выход из цикла Do

Для того чтобы выйти из цикла Do где-либо внутри его тела, воспользуйтесь инструкцией Exit Do, например:

```
Sub PrintLimitedBilling(curHourlyRate As Currency)
    ' Вывод таблицы стоимости работ для сумм, меньших 200 рублей
    Dim intHours As Integer
    intHours = 1
    Do
        Debug.Print intHours * curHourlyRate
        If intHours * curHourlyRate >= 200 Then
            Exit Do
        End If
        intHours = intHours + 1
    Loop Until intHours = 9
End Sub
```

Эта процедура работает идентично процедуре PrintBilling во всем, кроме того, что в ней дополнительно производится проверка на превышение суммы предела в 200 рублей. Если такой момент наступает, производится выход из цикла по оператору Exit Do и передача управления следующей за ним инструкции.

Безусловный переход GoTo

Последним оператором передачи управления, с которым необходимо познакомиться, является инструкция безусловного перехода GoTo. Вот пример ее использования в программе:

```
Sub CalculateBill(curRate As Currency, intHours As Integer)
    ' Вычисление суммы до определенного предела
    If intHours > 100 Then
        GoTo ExitHere
    End If
    Debug.Print curRate * intHours
```



```
ExitHere:  
End Sub
```

В данной процедуре `ExitHere` — это пример *метки*. Метка не является выполняемой строкой в языке VBA — это своеобразная закладка, по имени которой интерпретатор производит переход в тексте программы.

Инструкция `GoTo` передает управление оператору, следующему за указанной меткой. В приведенном примере если переменная `intHours` превышает значение 100, производится выход из процедуры.

Эффективность использования оператора `GoTo` долгое время оставалась предметом споров. Большинство профессионалов склонялись к предубеждению, что инструкции `GoTo` усложняют читабельность текста программы и делают менее прозрачной его структуру, поскольку читателю приходится резко перескакивать к другим фрагментам программы, иногда сильно разнесенным в тексте. Чаще всего фрагмент программы, содержащий оператор безусловного перехода, можно переписать так, чтобы исключить его использование. При этом процедура, приведенная выше, может иметь следующий вид:

```
Sub CalculateBill(curRate As Currency, intHours As Integer)  
    ' Вычисление суммы до определенного предела  
    If intHours <= 100 Then  
        Debug.Print curRate * intHours  
    End If  
End Sub
```

Результат работы этой функции ничем не отличается от предыдущей, однако здесь уже отсутствует оператор `GoTo`.

В целом согласимся с теми, кто старается избегать использования инструкции `GoTo`. Иногда может оказаться, что этот оператор делает программу более понятной, и только тогда вы вправе его использовать. К примеру, если в нескольких местах сложной процедуры нужно выполнить операции очистки переменных (или ликвидации объектов), а затем завершить процедуру. Здесь, во избежание многократного повторения одной и той же последовательности инструкций очистки, можно вполне обоснованно использовать оператор `GoTo`. Однако не считайте этот оператор панацеей на все случаи жизни.

➔ Оператор `GoTo` может быть составной частью предложения обработки ошибок `On Error Goto`, не следует избегать его использования в этих конструкциях. Более подробно об этой конструкции речь шла в главе 4.

ПРАКТИКУМ

Формирование таблицы стоимости почасовых работ

Консультанты нередко сталкиваются с задачей оценки объема проделанной ими работы. Довольно часто они вовлечены в несколько проектов, затрачивая на каждый из них по несколько часов. В этом случае полезно иметь под рукой таблицу, в которой явно указано, какой счет должен быть выставлен, скажем, за работу по три часа в день в течение четырех дней по ставке 25 рублей в час.

Для генерирования такой таблицы была создана процедура PrintBillingChart, в которой использованы некоторые из рассмотренных в настоящей главе инструкций передачи управления. Вот ее полный текст:

```
Sub PrintBillingChart(curBaseRate As Currency, _
                    intMaxDays As Integer)
    ' Создание таблицы сумм оплаты за почасовую работу
    ' в пределах заданного количества дней
    Dim intDays As Integer
    Dim intHours As Integer

    If intMaxDays > 6 Then
        Debug.Print "Эта процедура ограничена 6-ю днями"
    Else
        ' Создание заголовка
        Debug.Print "Почасовая ставка "&
            Format(curBaseRate, "Currency")

        ' Создание шапки таблицы
        Debug.Print vbTab;
        intDays = 0
        Do Until intDays = intMaxDays
            intDays = intDays + 1
            Debug.Print CStr(intDays) & "дней" & vbTab;
        Loop
        Debug.Print

        ' Формирование самой таблицы
        For intHours = 1 To 8
            Debug.Print CStr(intHours) & vbTab;
            intDays = 0
            Do Until intDays = intMaxDays
                intDays = intDays + 1
                Debug.Print Format(intDays * intHours * curBaseRate,
                    "Currency") & vbTab;
            Loop
            Debug.Print
        Next intHours
    End If
End Sub
```

Перед тем как разобрать эту программу, посмотрим на результат ее работы в окне Immediate при вызове PrintBillingChart 25, 4:

	Почасовая ставка 25,00р.			
	1 дней	2 дней	3 дней	4 дней
1	25,00р.	50,00р.	75,00р.	100,00р.
2	50,00р.	100,00р.	150,00р.	200,00р.
3	75,00р.	150,00р.	225,00р.	300,00р.
4	100,00р.	200,00р.	300,00р.	400,00р.
5	125,00р.	250,00р.	375,00р.	500,00р.
6	150,00р.	300,00р.	450,00р.	600,00р.
7	175,00р.	350,00р.	525,00р.	700,00р.
8	200,00р.	400,00р.	600,00р.	800,00р.

Как следует из таблицы, четырехдневная работа по три часа в день со ставкой 25 рублей в час оценивается в 225 рублей.

Процедура начинается с объявления используемых переменных (не забывайте, что объявление *всех* используемых в процедуре переменных в едином блоке является одним из признаков профессионализма). После этого производится проверка ширины таблицы (исходя из количества дней, переданных в качестве аргумента). В данном примере для выхода из процедуры в случае превышения аргументом установленного лимита удобно использовать конструкцию `If...Else...End`.

В данной процедуре существует всего три логических фрагмента. Все они для ясности выделены соответствующими комментариями. В первом фрагменте выполняется печать заголовка таблицы с помощью инструкции `Debug.Print`.

При печати шапки таблицы был использован один элемент, который вы еще не встречали в этой книге. Обратите внимание, что некоторые строки `Debug.Print` заканчиваются точкой с запятой. Это указывает интерпретатору на то, что следующий оператор вывода на печать должен выполняться без перевода строки. В результате, при печати заголовка в одной строке выводятся все фразы, после чего инструкция `Debug.Print` без аргументов переводит строку. Также обратите внимание на встроенную константу `vbTab`, представляющую собой символ табуляции.

При печати значений таблицы использовано вложение цикла `Do` в цикл `For...Next`. Таким образом, весь цикл `Do` выполняется восемь раз — по одному разу для каждой строки таблицы. В теле цикла `Do` происходит форматирование значения и вывод его на печать.

Несмотря на то, что данная процедура является более сложной, чем те, с которыми вы сталкивались в книге до сих пор, она составлена из тех же строительных блоков. Более внимательно присмотритесь к каждому из блоков, и вы увидите, как они складываются в единое целое. Если при этом у вас возникают сомнения, можете воспользоваться одним из режимов отладки — пошаговым или по контрольным точкам.

Массивы

7

Введение в переменные массивов

Все переменные, с которыми мы сталкивались до сих пор, имели достаточно простую структуру. Все они создавались для определенной цели, имели некоторый тип данных и соответствующие значения. Однако не всегда переменные столь просты — иногда они позволяют одновременно хранить несколько значений. В этой главе мы познакомимся с переменными массивов и методами хранения в них множеств значений.

Грубо говоря, массивы ссылаются на большие группы однородных значений. В математической терминологии массивом называют прямоугольное множество значений, структурированное по строкам и столбцам. В языке VBA *массив* — это множество фиксированных значений, называемых *элементами*, с общим именем и типом данных. Вся эта группа значений в VBA интерпретируется и обрабатывается как одна переменная.

Переменные массивов могут понадобиться в любой момент, когда необходимо хранение набора связанных значений в едином месте. К примеру, если в цикле многократно выполняется один и тот же набор инструкций и результаты вычислений каждого прохода требуется сохранить для последующей обработки, одним из вариантов организации таких данных явля-

В ЭТОЙ ГЛАВЕ

Введение в переменные массивов.....	135
Объявление переменных массивов.....	136
Понятие индекса массива	136
Работа с элементами массивов.....	138
Многомерные массивы	140
Динамические массивы.....	141

ется временный массив. Для того чтобы создавать, обслуживать и удалять временные массивы, необходимы дополнительные инструкции программы. Когда речь заходит о хранении множества связанных однотипных значений, средством решения задачи обычно являются массивы.

Объявление переменных массивов

Массив — это такая же переменная, как и все остальные, поэтому при его определении нужно указать тип данных. При определении массива следует указывать его нижнюю и верхнюю границу — другими словами, минимальный и максимальный номер ячейки.

При объявлении переменной массива используют следующий синтаксис:

```
Dim переменная_массива ([нижняя_граница] to верхняя_граница) _  
    As тип_данных
```

где *переменная_массива* — имя переменной; *нижняя_граница* и *верхняя_граница* — соответственно, наименьшее и наибольшее значение индекса; *тип_данных* — тип данных, которые могут храниться в массиве. Если при объявлении не указывать тип данных явным образом, он будет интерпретироваться как *Variant*.

В своей простейшей форме индексы массивов начинают отсчет с нуля, и в этом случае при объявлении их переменных значение нижней границы можно опускать. Однако рекомендуется всегда при объявлении указывать явно как верхнюю, так и нижнюю границу индекса, если у вас нет веских оснований последнюю не упоминать.

Рассмотрим простой пример оператора объявления массива:

```
Dim arrInteger(1 to 4) As Integer
```

Об этом объявленном массиве мы можем сказать следующее:

- в переменной *arrInteger* может храниться вплоть до четырех значений;
- в переменной *arrInteger* могут храниться исключительно целочисленные значения (типа *Integer*);
- элементы массива *arrInteger* имеют номера 1, 2, 3 и 4.

Понятие индекса массива

Каким же образом интерпретатор VBA может отличать отдельные элементы массива друг от друга? Очень просто: в языке VBA каждому элементу массива присваивается *индекс*. Образно говоря, индексом является некоторый тип идентификационного значения каждого элемента.

Значения индексов в массиве зависят от заданных при объявлении *верхней и нижней границ*. К примеру, объявление

```
Dim arrInteger(1 to 4) As Integer
```

указывает на хранение четырех целочисленных значений. Для этого создаются четыре элемента, имеющих разные целочисленные значения индекса —

от единицы до четырех. Очевидно, что первый элемент будет иметь индекс 1, второй — 2 и т.д.

В табл. 7.1 показаны примеры объявления нескольких массивов, создаваемых для хранения четырех целочисленных значений, однако в каждом из случаев индексные значения будут отличаться. Как видите, нумерация элементов строго определяется заданными при объявлении верхней и нижней границами.

Таблица 7.1. Примеры объявления массивов

Объявление массива	Значения индекса
Dim arrInteger(0 to 3) As Integer	0, 1, 2, 3
Dim arrInteger(2 to 5) As Integer	2, 3, 4, 5
Dim arrInteger(-4 to -1) As Integer	-4, -3, -2, -1
Dim arrInteger(-3 to 0) As Integer	-3, -2, -1, 0
Dim arrInteger(3) As Integer	0, 1, 2, 3

Совет

Иногда значение, хранимое в элементе массива, может случайно совпасть (или иметь сходство) с его индексом, но не путайте эти два понятия. Индекс является средством обнаружения интерпретатором VBA места нахождения соответствующего значения относительно начального адреса массива в оперативной памяти¹. К примеру, пусть первый элемент массива может хранить некоторое целочисленное значение (скажем, 1, 17 или -257). Не пытайтесь искать связь между индексом элемента и его значением, если таковая изначально не была заложена в алгоритме (а такое случается исключительно редко).

Инструкция Option Base

По умолчанию нижняя граница массива принимается равной нулю. Исключение составляет любое из следующих условий:

- нижняя граница массива при объявлении указана явно;
- в модуле имеется инструкция Option Base.

¹ Все элементы массива размещаются в оперативной памяти последовательно. При этом каждому элементу отводится для хранения определяемое типом количество байт. Все переменные могут ссылаться только на один адрес в памяти, в том числе и переменные массивов, которые ссылаются на адрес первого элемента. Таким образом, чтобы найти третий по счету элемент массива, интерпретатор берет значение индекса, вычитает из него значение нижней границы и умножает на длину данного типа данных. В результате получится смещение адреса в памяти искомого элемента относительно адреса, хранимого в переменной массива. После сложения этих двух величин можно получить абсолютный адрес элемента в памяти и извлечь его значение. — Примеч. ред.

Вы уже видели первое условие в действии, так что мы остановимся сейчас только на инструкции `Option Base`. Эта инструкция вводится в области общих определений модуля (General Definition) в следующей форме:

```
Option Base 0 | 1
```

Эта инструкция устанавливает нижнюю границу, которая будет использоваться по умолчанию *в этом и только этом* модуле. В данной инструкции можно установить нижнюю границу в нуль или единицу, но ни в какое другое число.

➔ Область общих объявлений модуля была описана в разделе “Введение в модули VBA” главы 2.

Совет

Несмотря на то что инструкция `Option Base 1` автоматически устанавливает нижнюю границу всех массивов в единицу, старайтесь задавать эту границу при объявлении всех массивов явным образом — это повысит прозрачность текста программы. Это также поможет избежать потенциальных (и вполне вероятных) ошибок, скажем, если вы забудете, что нижняя граница четырехэлементного массива неявно установлена инструкцией `Option Base` не в 1, а в 0, и попытаетесь обратиться к элементу массива с индексом 4.

Работа с элементами массивов

После объявления переменной массива работать с его элементами можно двумя способами:

- можно их определять;
- можно на них ссылаться.

Определение и ссылка на элементы массива идентичны тем же операциям с обычными переменными, только в первом случае не следует забывать про значения индексов.

Определение элементов массива

Объявив переменную массива, вы сделали только полдела. Следующим шагом будет занесение в его элементы реальных значений. Для этого можно использовать следующий синтаксис:

```
переменная_массива(индекс) = значение
```

где *индекс* определяет положение элемента в массиве, а *значение* — любое допустимое значение или выражение, соответствующее объявленному типу данных. В приведенном выше примере

```
Dim arrInteger(1 to 4) As Integer
```

индекс при присвоении может принимать значения 1, 2, 3 и 4, а *значением* может быть любое целое число. В частности, будут допустимы все приведенные далее инструкции:

- `arrInteger(1) = 100`
- `arrInteger(2) = -200`
- `arrInteger(3) = 3`
- `arrInteger(4) = 40`

В этих инструкциях четырем элементам массива были присвоены значения 100, -200, 3 и 40 соответственно.

Аналогично, в следующей процедуре четырем элементам массива будут присвоены значения 1, 2, 3 и 4:

```
Public Sub ArrayExample1()
    Dim intCounter As Integer
    Dim arrExample(1 To 4) As Variant
    For intCounter = 1 To 4
        arrExample(intCounter) = intCounter * 0.1
    Next
End Sub
```

В данном массиве нижней границей является 1, а верхней — 4. В цикле `For...Next` переменной `intCounter` присваиваются последовательно значения от 1 до 4. Таким образом, при каждом выполнении цикла переменная `intCounter` соответствует одному из допустимых индексов массива `arrExample`. В процедуре значение индекса умножается на 0,1, после чего результат присваивается текущему элементу. В частности, при первом проходе переменная `intCounter` равна единице, так что определение элемента массива принимает следующий вид:

```
arrExample(1) = 1 * 0.1
```

или

```
arrExample(1) = .1
```

При втором прохождении цикла значением переменной `intCounter` становится двойка, так что элемент `arrExample(2)` будет равен 0,2 и т.д.

➔ Описание конструкции цикла `For...Next` содержится в соответствующем разделе главы 6.

Ссылка на элементы массива

В описанной выше процедуре каждому из четырех элементов массива присваивалось некоторое значение. Теперь нам нужно узнать, как обращаться к этим значениям. Эта процедура выполняется практически тем же способом, каким происходило и присвоение — путем использования индексов:

переменная = переменная_массива(индекс)

В частности, для обращения к любому элементу массива `arrExample`, с которым мы работали в предыдущем разделе, нужно использовать его индекс:

```
arrExample(1)
arrExample(2)
arrExample(3)
arrExample(4)
```


Для иллюстрации обращения к элементам массива приведем следующую процедуру, которой передается в качестве аргумента индекс нужного элемента, а она выводит его на печать.

```
Public Sub ArrayExample2(ele As Integer)
    Dim intCounter As Integer
    Dim arrExample(1 To 4) As Variant
    For intCounter = 1 To 4
        arrExample(intCounter) = intCounter * 0.1
    Next
    MsgBox "Значением элемента массива "& _
        ele & "является "& arrExample(ele), vbOKOnly
End Sub
```

Чтобы увидеть эту процедуру в действии, запустите редактор VBE, откройте пустой модуль и введите в него приведенный выше текст (эта процедура уже введена в модуль Chapter7 базы данных примеров TimeTrack.mdb). После этого в окне Immediate введите следующую инструкцию:

arrayExample2(x)

где x принимает значения 1, 2, 3 или 4, после чего нажмите клавишу <Enter>. (В этой процедуре не заложена обработка ошибок, поэтому в случае передачи

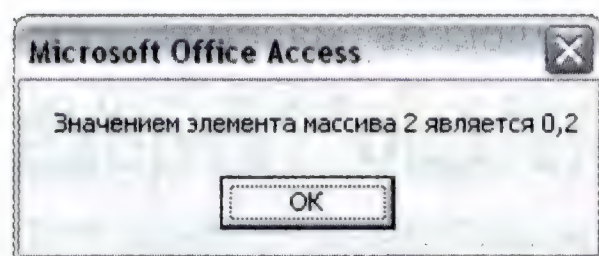


Рис. 7.1. Для ссылки на любой из элементов массива используется его индекс

ей значения, выпадающего из диапазона от единицы до четырех, в окне будет выведено неполное сообщение.) В процедуре всем четырем элементам массива присваиваются значения, но на печать выводится только один из них. На рис. 7.1 показан результат передачи процедуре аргумента 2. Щелкните на кнопке ОК, закрывая окно сообщения, и вернитесь в редактор VBE.

➔ Если вы хотите узнать, как включать процедуру обработки ошибок, обратитесь к материалу соответствующего раздела главы 4.

Многомерные массивы

Для обращения к элементу в массивах может использоваться несколько индексов; в частности, в двумерном массиве для этого используется два индекса. В объявлении двумерного массива задаются диапазоны обоих индексов, разделенные запятыми. К примеру, в следующей инструкции объявляется массив размерности 10×10:

```
Dim arrTaxAmount(1 To 10, 1 To 10) As Double
```

При определении и извлечении значений элементов двумерного массива используются оба индекса, например:

```
Public Function GetTaxAmount(intPercent As Integer, _
    intAmount As Integer) As Double
    ' Возвращает сумму налогов для заданных
    ' процента налога и суммы покупки
    ' Оба аргумента ограничены диапазоном от 1 до 10
    Dim i As Integer
```



```

Dim j As Integer
Dim arrTaxAmount(1 To 10, 1 To 10) As Double
' Построение массива
For i = 1 To 10
    For j = 1 To 10
        arrTaxAmount(i, j) = (i * j) / 100
    Next j
Next i

' Извлечение из массива суммы налогов
GetTaxAmount = arrTaxAmount(intPercent, intAmount)
End Function

```

Массивы могут иметь до 60 измерений — это намного превышает любые потенциальные потребности программистов.

Динамические массивы

До сих пор во всех примерах при объявлении массивов мы явным образом задавали верхнюю и нижнюю границы. Таким образом мы создавали *массивы фиксированной длины*. Если при объявлении эти границы не указывать явно, будет создан *динамический массив*. После объявления такого массива для изменения его длины используют инструкцию `ReDim` следующим образом:

```

Dim arrExample() As тип_данных
...
ReDim arrExample([нижняя_граница] To верхняя_граница) As тип_данных

```

Так как язык VBA все же настаивает на явном задании границ массива, описанный выше метод является единственным средством управления значениями верхней и нижней границ.

Инструкция ReDim

Инструкция `ReDim` изменяет размерности динамического массива и перераспределяет пространство, под него выделенное. С помощью этой инструкции после объявления массива, но перед первым его использованием задаются размерности.

Инструкция `ReDim` имеет следующий синтаксис:

```

ReDim [Preserve] переменная_массива ([нижняя_граница] To _
верхняя_граница) As тип_данных

```

где ключевое слово `Preserve` указывает на сохранение данных массива при изменении только последнего измерения (т.е. его границ). Если это ключевое слово не указывать, существующие значения массива стираются.

В следующей процедуре используется динамический массив:

```

Public Sub ArrayExample3(l As Integer, u As Integer)
    Dim intCounter As Integer
    Dim arrExample() As Variant
    ReDim arrExample(1 To u)
    For intCounter = 1 To u
        arrExample(intCounter) = intCounter * 0.1
    Next

```



```

    For intCounter = 1 To u
        Debug.Print arrExample(intCounter)
    Next
End Sub

```

В этой процедуре при объявлении массива явным образом не указываются границы, и в то же время это компенсируется наличием инструкции `ReDim`. При вызове этой процедуры ей можно передавать любые значения нижней и верхней границ массива, что делает ее значительно более гибкой.

Эту процедуру вы можете набрать самостоятельно в редакторе VBE в окне нового модуля или воспользоваться модулем `Chapter7` базы данных примеров `TimeTrack.mdb`. В любом случае вызовите эту процедуру в окне `Immediate`, набрав следующую инструкцию:

```
ArrayExample3 1, 4
```

На рис. 7.2 показан результат выполнения этой процедуры — числа 0,1, 0,2, 0,3 и 0,4. Теперь попробуйте вызвать эту же функцию, передавая ей аргументы 2 и 6, — результат показан на рис. 7.3.

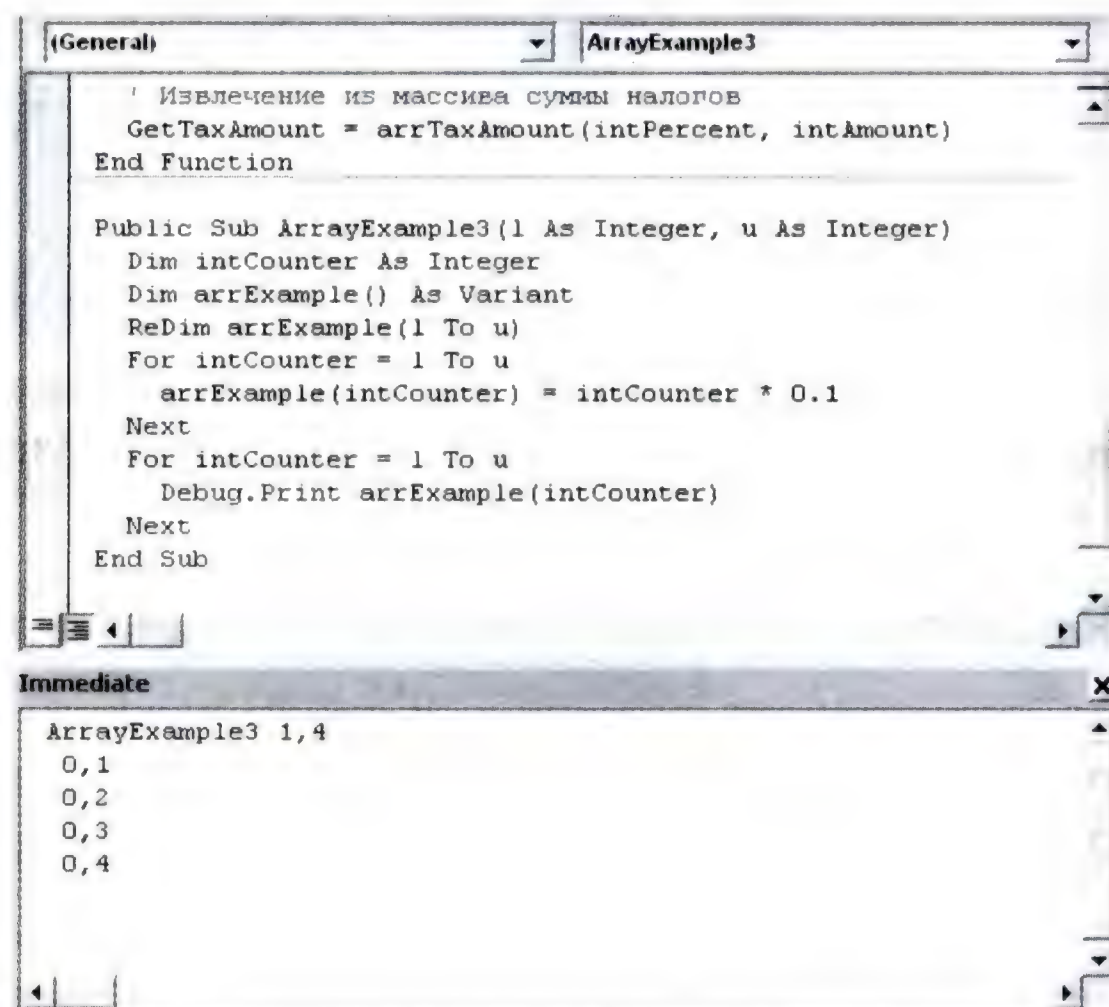


Рис. 7.2. Передайте нижнюю и верхнюю границы в качестве аргументов процедуры



Рис. 7.3. Изменяйте размерности массива, передавая в качестве аргументов разные значения

Объекты

8

Понятие объекта

До сих пор в книге мы встречались только с несколькими переменными. Вспомним, что *переменной* называется именованный элемент кода VBA, хранящий некоторое значение (например, число или символьную строку). В случае переменной массива в ней может храниться множество однотипных значений. В то же время, все эти переменные принадлежали одному типу — *простые переменные* (иногда в технической литературе можно встретить и другое название — *скалярные переменные*). Однако существует и еще один тип — *переменные объектов*, или просто *объекты*. Переменные этого типа наряду со значениями хранят также и некоторый тип поведения. Эту концепцию начинающим иногда тяжело уловить сразу, поэтому при объяснении материала о работе объектов в настоящей главе будут использованы аналогии.

Ссылка на реальный мир

Чтобы понять концепцию объектов в программировании, нужно вначале взглянуть на объекты реального мира. В настоящем разделе мы остановимся на радиоуправляемых моделях машин, которые так популярны среди детей (и программистов!). Такая игрушка обладает тремя ключевыми особенностями.

В ЭТОЙ ГЛАВЕ

Понятие объекта.....	143
Чтение и установка свойств	146
Вызов методов	148
Работа с коллекциями.....	150
Объектная модель	152
Создание собственных объектов	155
Работа с событиями.....	158

- Каждая из моделей выпускается массово, т.е. существует великое множество машинок одного типа.
- Каждая конкретная машинка имеет множество характеристик, таких как цвет. Разные машинки могут иметь разные цвета.
- При манипуляциях с рычажками пульта управления такая машина выполняет определенные движения: например, едет вперед или назад, поворачивает влево или вправо. Вы можете не знать, с помощью каких механизмов эти движения выполняются — ваша задача заключается только в подаче соответствующих команд.

При изложении материала об объектах мы будем обращаться к аналогии с этими машинками, это поможет вам легко понять ключевые концепции.

Пример объекта Access

В языке VBA и приложениях, так же как и в реальном мире, существуют объекты, которыми можно манипулировать. К примеру, любая форма Access на самом деле является объектом. Вот как работает эта аналогия.

- Объекты массово производятся из некоторого шаблона, называемого *классом*. Каждый объект, произведенный из шаблона, называется *экземпляром класса*. Таким образом, один класс может произвести на свет множество объектов.
- Любая форма описывается набором *свойств*, такими как, например, название “Переключатель”. Все формы имеют свойство с названием `Caption`, но разные формы могут иметь разные значения свойства.
- Форма может выполнять некоторые действия, к примеру, открываться и закрываться. Вам совершенно не требуется знать, как именно реализованы в форме эти операции — достаточно вызвать соответствующий *метод*.

Естественно, не всем аспектам объектов можно привести аналогии: существуют и такие (к примеру, события, о них речь пойдет далее), которым в реальном мире подыскать аналогию сложно. В то же время, если вам трудно представить некоторые концепции объектной технологии в программировании, вспомните об объектах реального мира — это вам поможет.

Создание объектов в тексте программы

Несмотря на то, что отдельные объекты (такие как формы) можно создать и с помощью интерфейса пользователя Access, большинство объектов представляют собой совершенно абстрактные вещи и могут создаваться только программным способом. Перед тем как проводить с объектом какие-либо манипуляции, его нужно создать. Вот пример кода VBA, в котором создается новый объект — на этот раз экземпляр формы `Projects`, которая описана в базе данных примеров `TimeTrack.mdb`.

```
Sub CreateObject1()  
    ' Создается и отображается объект формы
```



```

Dim frm As Form_Projects
Set frm = New Form_Projects
frm.Visible = True
MsgBox "Для продолжения щелкните ОК"
End Sub

```

Если запустить на выполнение эту программу, откроется экземпляр формы Project и окно сообщения (рис. 8.1). Если щелкнуть на кнопке ОК, как окно сообщения, так и сама форма закроются.

Текст процедуры начинается с объявления переменной объекта с именем frm. Эта переменная не имеет простого типа, такого как Integer или String — при объявлении указан тип Form_Project. В Access для каждой формы создается свой класс, носящий название Form_ИмяФормы.

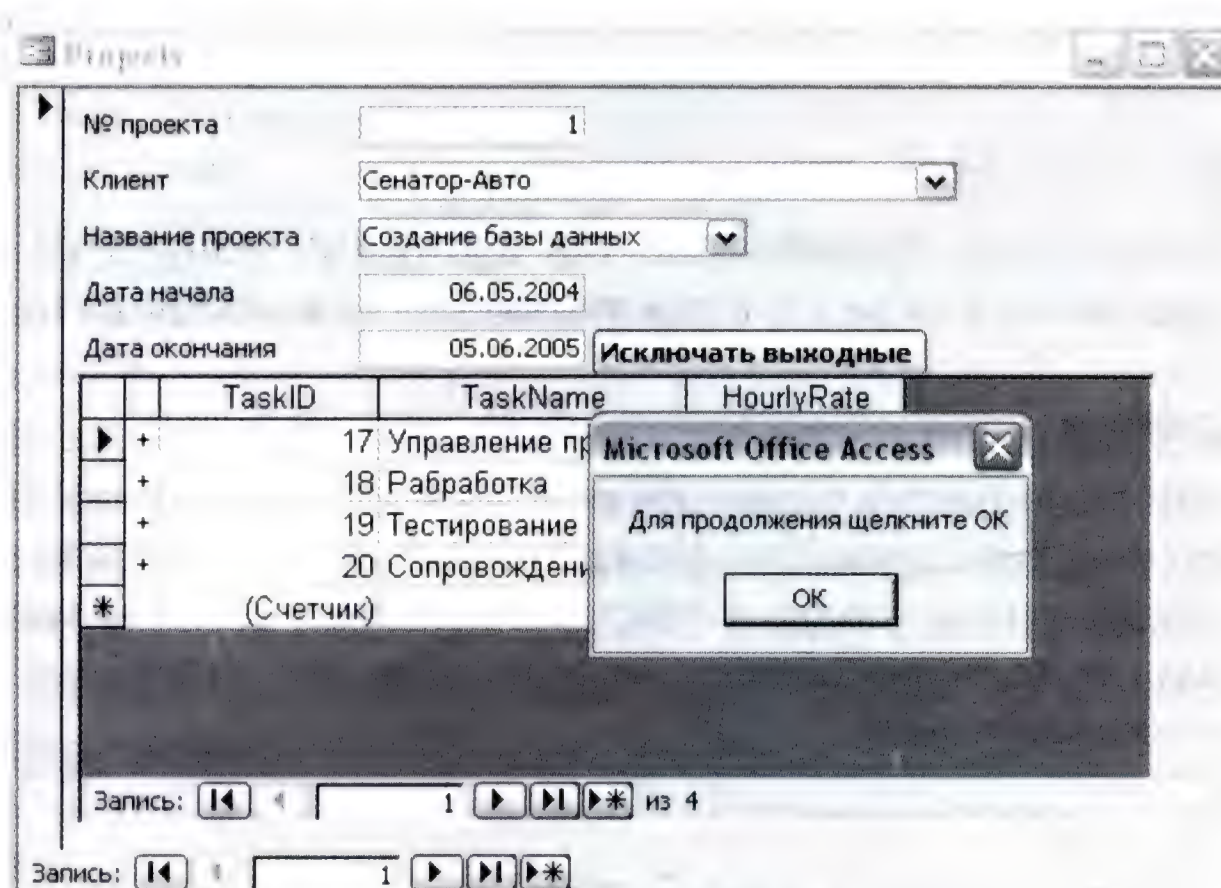


Рис. 8.1. Объект формы, созданный в коде VBA

Следует помнить, что класс сам по себе не является объектом. Для манипуляции объектом нужно создать экземпляр класса. Именно для этой цели предназначена строка с инструкцией Set. В этой строке интерпретатору VBA дается указание на создание экземпляра объекта класса Form_Project, ссылка на который присваивается переменной frm. Именно по этой ссылке объект будет использоваться в дальнейшем.

По умолчанию в Access все создаваемые формы изначально являются скрытыми (т.е. невидимыми). В следующей строке процедуры свойству видимости Visible присваивается значение True, что приводит к отображению формы на экране.

Далее в программе следует инструкция MsgBox, отображающая окно сообщения. Не забывайте, что при открытии окна сообщения выполнение программы приостанавливается до тех пор, пока пользователь на него не отреагирует нажатием какой-либо кнопки.

После щелчка на кнопке ОК форма с экрана исчезает. Это происходит потому, что интерпретатор VBA уничтожил все переменные, объявленные в про-

цедуре, после ее завершения. После того как переменная `frm` уничтожена, отображение соответствующей формы на экране прекращается.

- ➔ Можно объявлять переменные и несколько иным способом, позволяющим их использовать и вне данной процедуры. Более подробно вопросы области определения переменных описаны в главе 9.

Существует еще один способ создания объектов, требующий меньший объем текста:

```
Sub CreateObject2()  
    ' Создается и отображается объект формы  
    Dim frm As New Form_Projects  
    frm.Visible = True  
    MsgBox "Для продолжения щелкните ОК"  
End Sub
```

В строке

```
Dim frm As New Form_Projects
```

переменная объекта как объявляется, так и инициализируется (т.е. создается соответствующий новый объект) с помощью ключевого слова `New`.

Предупреждение

При объявлении и инициализации объекта в одной инструкции на самом деле новый объект еще не создается. Он создается в тот момент, когда переменная объекта впервые будет использована в тексте программы. Если вы хотите прояснить момент создания экземпляра объекта, используйте две отдельные инструкции — объявления и инициализации.

Чтение и установка свойств

Свойства объекта описывают его текущее состояние. Любой класс может как не иметь свойств, так и иметь их большое количество. При создании объекта из класса всем его свойствам присваиваются некоторые значения. Программным путем можно извлечь значения всех свойств объекта, а также присвоить им новые. Вот пример процедуры, извлекающей текущие значения свойств объекта формы:

```
Sub ReadProperties()  
    ' Отображение некоторых свойств объекта  
    Dim frm As New Form_Projects  
    frm.Visible = True  
    Debug.Print frm.Caption  
    Debug.Print frm.RecordSelectors  
End Sub
```

Если запустить эту процедуру в окне `Immediate`, на экран будет выведено следующее:

```
Projects  
True
```


Для чтения значения любого свойства объекта нужно обратиться к имени объекта, дополненному через точку именем свойства:

имя_объекта.имя_свойства

Эту переменную можно использовать для извлечения и присвоения значения, а также для передачи аргумента некоторой процедуре. Грубо говоря, эту переменную можно использовать везде, где можно использовать простую переменную того же типа. В данном случае название формы — `Projects`, а полоса перехода между записями (свойство `RecordSelectors`), расположенная на левой границе окна, включена.

Для изменения свойства ему просто нужно присвоить новое значение, используя очень похожий синтаксис:

```
Sub WriteProperties()  
    ' Изменение некоторых свойств объекта  
    Dim frm As New Form_Projects  
    frm.Visible = True  
    frm.Caption = "Характеристики проекта"  
    frm.RecordSelectors = False  
    MsgBox "Для продолжения щелкните ОК"  
End Sub
```

При запуске этой процедуры отобразится результат, подобный показанному на рис. 8.2. Обратите внимание, что название формы установлено в значение, определенное в программе, а полоса перехода между записями уже не отображается. (Сравните рис. 8.1 с рис. 8.2 и вы увидите, что из верхнего левого угла последней формы исчез маленький черный треугольник. Это и есть индикатор выбора записи.)

Примечание

Если после запуска процедуры `WriteProperties` открыть эту же форму в интерфейсе пользователя Access, обнаружится, что все изменения, сделанные в коде VBA, аннулированы. По умолчанию все изменения, выполненные в коде VBA, сбрасываются, когда работа с объектом прекращается. Это поведение по умолчанию можно изменить, вызвав метод `Save` данной формы. Более подробно о методах вы узнаете в следующем разделе этой главы.

Любой объект может иметь три типа свойств:

- доступные для чтения и изменения в программе;
- доступные в программе только для чтения, но не для изменения;
- доступные в программе только для изменения, но не для чтения.

Большинство свойств доступно и для чтения, и для записи. Доступными только для чтения свойствами являются, как правило, те, которые нельзя изменять после создания объекта. В нашем примере объекта `Form` таким свойством является название класса — `Name`. Доступные только для записи свойства встречаются крайне редко. К примеру, для ограничения доступа к

объекту можно дополнить его доступным только для записи свойством пароля. При этом пароль можно будет только установить, но не увидеть.

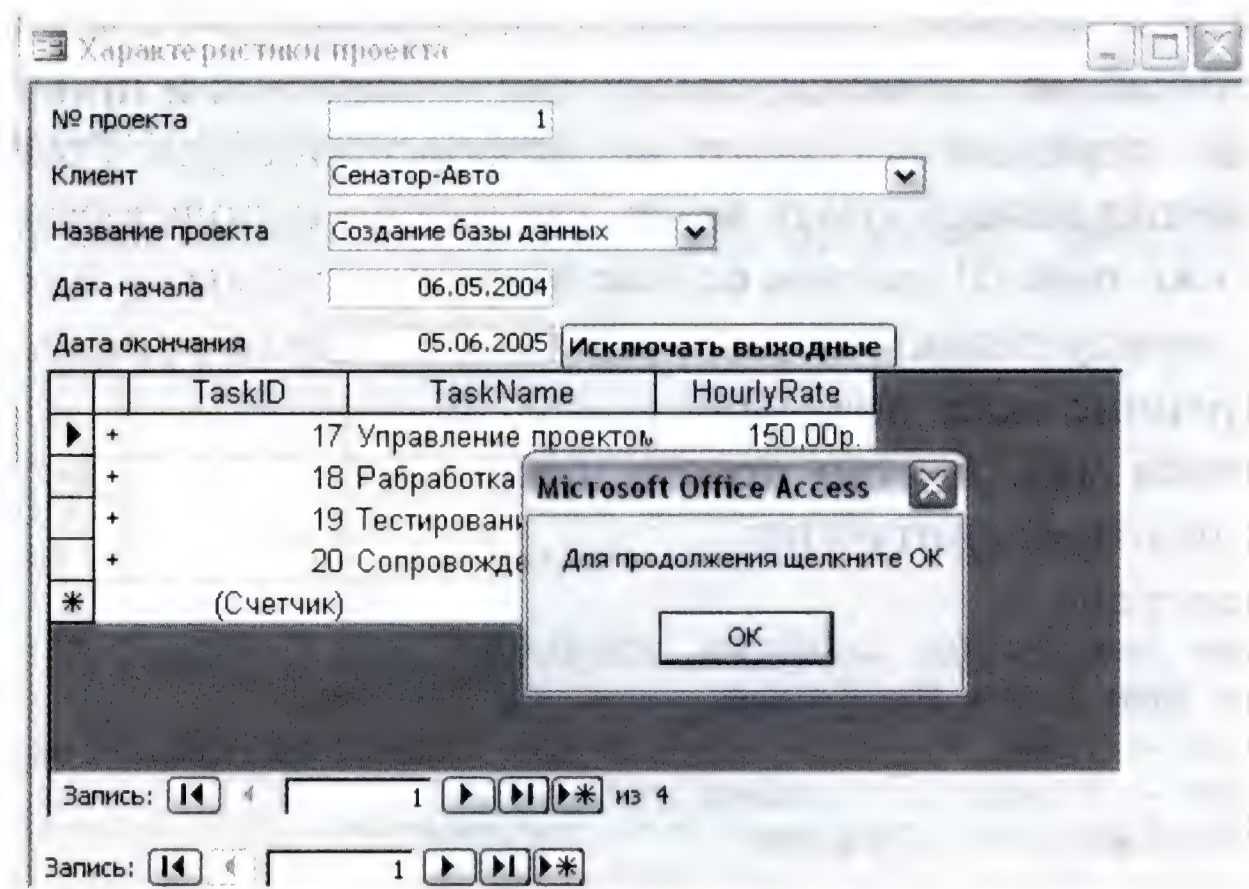


Рис. 8.2. Изменение значений свойств объекта в VBA

Вызов методов

Действия, которые может выполнять объект, называют его *методами*. Для вызова метода используется синтаксис, аналогичный установке свойств объекта. Имя метода отделяется от имени объекта символом точки. Вот пример вызова некоторых методов формы:

```
Sub InvokeMethod()  
    ' Вызов методов объекта  
    Dim frm As New Form_Projects  
    frm.Visible = True  
    frm.Move 0, 0, 1440, 1440  
    MsgBox "Для продолжения щелкните ОК"  
End Sub
```

В этой процедуре вызывается метод `Move` (Перемещение) класса `Form`. Методы могут иметь свои аргументы, следующие при вызове после их названия, и разделенные запятыми. В данном случае при вызове метода `Move` было передано четыре аргумента:

- первый аргумент указывает на новое положение верхнего левого угла окна по горизонтали (относительно верхнего левого угла рабочей области Access);
- второй аргумент указывает на новое положение верхнего левого угла окна по вертикали (также относительно верхнего левого угла рабочей области Access);
- третий и четвертый аргументы содержат новые ширину и высоту формы соответственно.

На рис. 8.3 показан результат запуска процедуры `InvokeMethod`. Окно формы сдвинулось к верхнему левому углу рабочей области приложения, а ее высота и ширина уменьшились и стали равными.

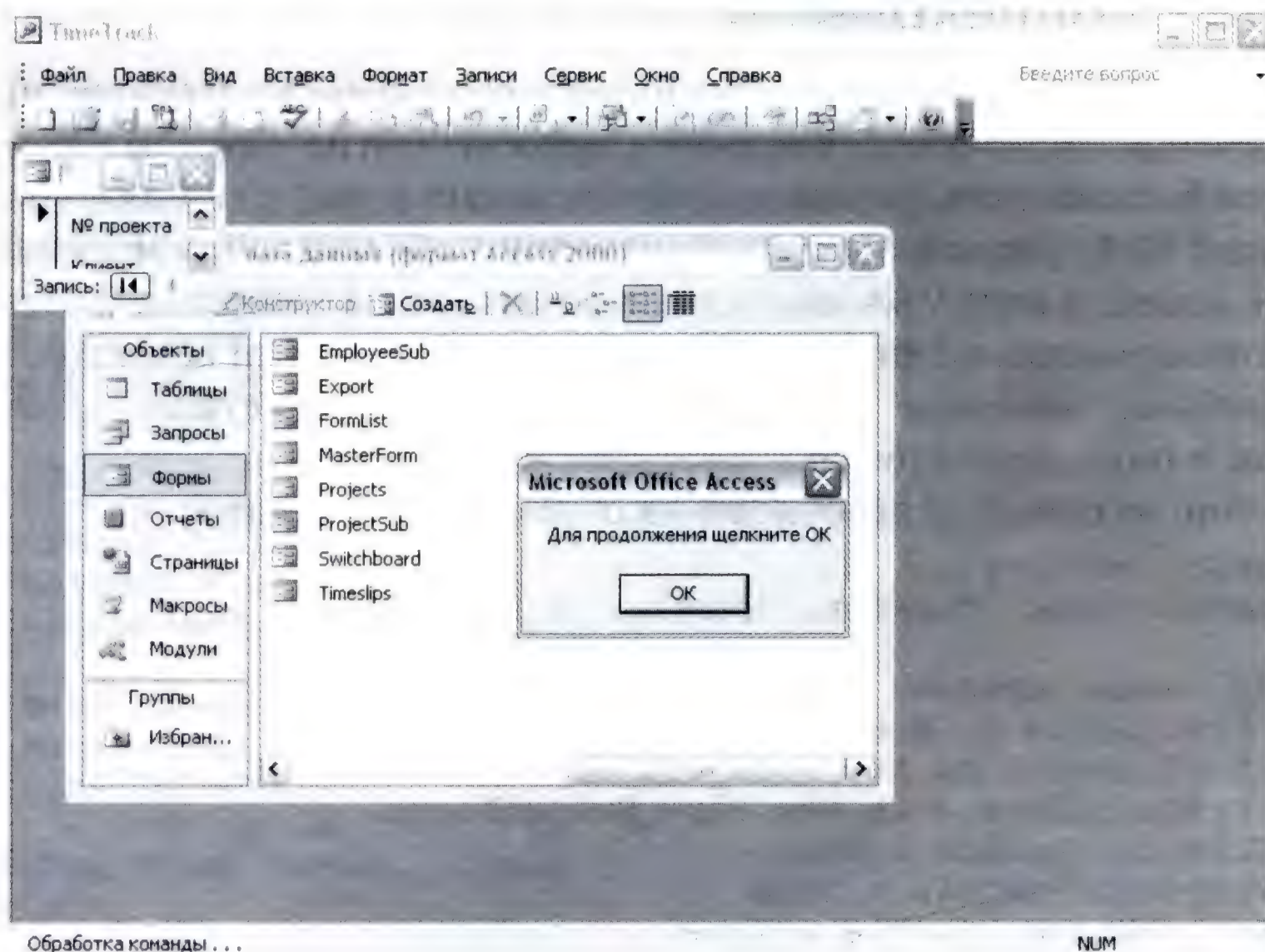


Рис. 8.3. Форма после вызова метода `Move`

Примечание

В языке VBA единицей измерения расстояний и размеров является *твип*¹, численно равный 1/1440 дюйма.

Многие методы имеют необязательные аргументы. На самом деле последние три аргумента метода `Move` не обязательны. Если не передать какой-либо из этих аргументов, соответствующее свойство формы не изменится. Чтобы опустить аргументы, достаточно между соответствующими запятыми при вызове ничего не вставлять. К примеру, если бы требовалось изменить только положение границы окна по горизонтали и его ширину, можно было бы использовать следующую строку программы:

```
frm.Move 0, , 1440
```

Таким образом, чтобы исключить необязательные аргументы в конце списка параметров, можно даже не вводить при вызове соответствующие запятые. К аргументам можно обратиться также по имени — в этом случае их порядок при вызове метода значения не имеет, например:

¹ Твип (англ. twip) — типографская единица измерения, равная одной двадцатой пункта (point), отсюда и название. — *Примеч. ред.*


```
frm.Move Width:=1440, Top:=0
```

Оператор `:=` отделяет имя аргумента от его значения.

Работа с коллекциями

Несколько объектов можно объединить в *коллекции*. Коллекцией называют объект специального типа, который содержит другие объекты. К примеру, Access создает собственную коллекцию открытых форм, и ее можно использовать в коде VBA. Так как обслуживанием этого объекта занимается само приложение Access, в коде VBA инициализировать его не требуется. В этом разделе вы познакомитесь с синтаксисом работы с коллекциями объектов. Коллекции наиболее полезны в операциях поиска конкретных экземпляров объектов, в них содержащихся.

Вот один из способов перечисления объектов в коллекции:

```
Sub ListCollection1()  
    ' Перечисление объектов, хранимых в коллекции  
  
    ' Для начала откроем три формы  
    Dim frmClients As New Form_Clients  
    Dim frmProjects As New Form_Projects  
    Dim frmSwitchboard As New Form_Switchboard  
    frmClients.Visible = True  
    frmProjects.Visible = True  
    frmSwitchboard.Visible = True  
  
    ' Теперь пройдемся по коллекции форм  
    ' и выведем на печать их имена  
    Dim i As Integer  
    For i = 0 To Forms.Count - 1  
        Debug.Print Forms.Item(i).Name  
    Next i  
  
End Sub
```

Эта процедура начинает свою работу с открытия трех форм; при этом используется однострочный синтаксис объявления и инициализации объекта. После этого объявляется целочисленная переменная `i`, которая будет выступать счетчиком цикла. В цикле переменная `i` проходит значения от нуля до `Forms.Count` (это свойство, содержащее количество объектов во встроенной коллекции `Forms`). Нумерация объектов в коллекции начинается с нуля. Это значит, что номерами объектов в коллекции с тремя элементами будут 0, 1 и 2.

➔ Использованию циклов был посвящен раздел “Конструкция цикла `For...Next`” главы 6.

Каждая итерация в цикле возвращает имя одного из объектов, содержащихся в коллекции. Свойство `Item` коллекции возвращает указатель на соответствующий объект (к примеру, `Forms.Item(1)` возвращает второй объект коллекции). Так как в данной коллекции содержатся формы, то и элементы объекта являются объектами форм. Следовательно, мы можем обратиться

к свойству Name и извлечь имя формы. После запуска этой процедуры в окне Immediate появится следующий результат:

```
Clients
Projects
Switchboard
```

Существует два способа упрощения этой программы. Во-первых, свойство Item является *свойством по умолчанию* коллекции Forms. При обращении к такому свойству в текст программы не обязательно вставлять его имя. Таким образом, следующая процедура будет продолжать перечислять открытые формы:

```
Sub ListCollection2()
    ' Перечисление объектов, хранимых в коллекции
    ' Для начала откроем три формы
    Dim frmClients As New Form_Clients
    Dim frmProjects As New Form_Projects
    Dim frmSwitchboard As New Form_Switchboard
    frmClients.Visible = True
    frmProjects.Visible = True
    frmSwitchboard.Visible = True

    ' Теперь пройдемся по коллекции форм
    ' и выведем на печать их имена
    Dim i As Integer
    For i = 0 To Forms.Count - 1
        Debug.Print Forms(i).Name
    Next i

End Sub
```

В заключение, так как цикл проходит по *всем* элементам коллекции, то можно использовать специальную конструкцию, созданную именно для этих целей. Цикл For...Each проходит по всему содержимому коллекции, присваивая переменной frm ссылку на каждый из объектов, в нем содержащийся.

```
Sub ListCollection3()
    ' Перечисление объектов, хранимых в коллекции
    ' Для начала откроем три формы
    Dim frmClients As New Form_Clients
    Dim frmProjects As New Form_Projects
    Dim frmSwitchboard As New Form_Switchboard
    frmClients.Visible = True
    frmProjects.Visible = True
    frmSwitchboard.Visible = True

    ' Теперь пройдемся по коллекции форм
    ' и выведем на печать их имена
    Dim frm As Form
    For Each frm In Forms
        Debug.Print frm.Name
    Next frm

End Sub
```

При использовании конструкции For...Each переменная счетчика цикла должна иметь тип объекта соответствующего типа. Когда интерпретатор VBA

проходит по инструкциям цикла, он иницирует эту переменную, устанавливая в ней ссылку последовательно на все объекты коллекции. Если вы используете этот синтаксис, то можете не волноваться о значении свойства Count (Количество объектов) — об этих мелочах позаботится сам интерпретатор VBA.

Любая коллекция имеет свойство Count и метод Item. Во многих коллекциях вы можете найти также методы Add и Remove, предназначенные для добавления и удаления объектов в коллекции. Так как коллекция Forms управляется самим приложением Access, методы ручного добавления и удаления в ней отсутствуют.

Объектная модель

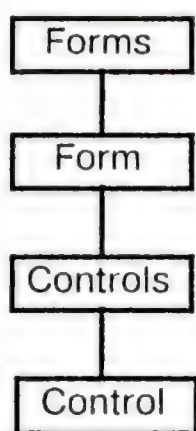


Рис. 8.4. Фрагмент объектной модели приложения Access

Коллекции и все элементы, которые они содержат, часто организуются в *объектную модель*. Объектной моделью называют диаграмму, показывающую связь между отдельными объектами в приложении. В качестве примера на рис. 8.4 показан фрагмент объектной модели, поддерживаемой программой Access.

На этой диаграмме мы видим, что коллекция Forms содержит отдельные объекты Form. В свою очередь, каждый из объектов Form содержит коллекцию Controls (Элементы управления), в которой хранится набор объектов Control. Как вы, наверное, уже догадались, каждый из объектов Control описывает некоторый элемент управления на форме.

Использование объектной модели

Вот небольшая процедура, использующая показанный выше фрагмент объектной модели:

```

Sub ListControls()
    ' Перечисление элементов управления
    ' Всех открытых форм

    ' для начала открываем три формы
    Dim frmClients As New Form_Clients
    Dim frmProjects As New Form_Projects
    Dim frmSwitchboard As New Form_Switchboard
    frmClients.Visible = True
    frmProjects.Visible = True
    frmSwitchboard.Visible = True

    ' Используя коллекции,
    ' перечисляем элементы управления форм
    Dim frm As Form
    Dim cnt As Control
    For Each frm In Forms
        Debug.Print frm.Name
        For Each cnt In frm.Controls
  
```



```

        Debug.Print " "& cnt.Name
    Next cnt
Next frm

```

```
End Sub
```

В этой процедуре вложенные циклы `For...Each` проходят по двум уровням объектной модели — коллекции форм и коллекции элементов управления. В результате в список попадут имена всех элементов управления всех трех открытых форм.

Использование ссылок

Многие приложения предоставляют в распоряжение пользователя свои объектные модели, и их можно использовать в коде VBA. К примеру, все приложения пакета Microsoft Office имеют собственные объектные модели. Для их использования из приложения, отличного от того, в котором запущена программа на языке VBA, в первую очередь нужно установить *ссылки* на соответствующие объектные модели. Само существование ссылки указывает интерпретатору VBA на то, что вы собираетесь использовать объекты данной объектной модели.

Для того чтобы установить новую ссылку, в редакторе VBA выберите пункт меню `Tools⇒References`, в результате чего откроется диалоговое окно, показанное на рис. 8.5. Ссылки, которые в текущем проекте VBA уже установлены, перечислены в верхней части списка окна, после чего в алфавитном порядке перечислены ссылки, доступные на данном компьютере.

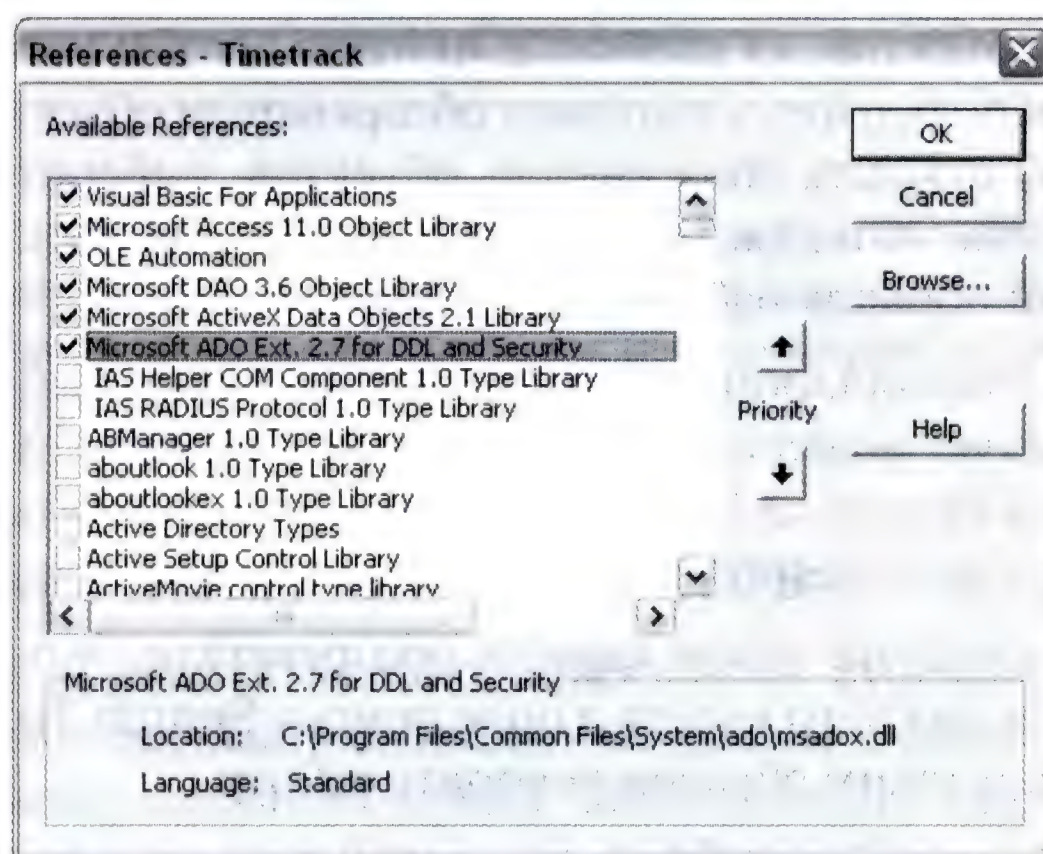


Рис. 8.5. Диалоговое окно ссылок

Чтобы включить ссылку на некоторую объектную модель, поставьте флажок в ее строке (при необходимости вы можете воспользоваться полосами прокрутки) и щелкните на кнопке `ОК`. После того как на объектную модель установлена ссылка, ее объекты вы можете использовать в своих программах.

К примеру, если установлена ссылка на библиотеку объектов Excel 11.0, следующая процедура успешно будет выполнена и в приложении Access:

```
Sub TestExcel()  
    ' Использование объектной модели Excel  
    Dim objExcel As New Excel.Application  
    objExcel.Visible = True  
    MsgBox "Excel should be visible"  
End Sub
```

Так как оба приложения — Excel и Access — имеют объекты с именем `Application`, в программе необходимо явно указать, какой именно объект мы собираемся использовать. Для этого при указании типа переменной в инструкции ее объявления перед именем объекта через точку указывается имя самого приложения, к примеру:

`Access.Form`

`Excel.Application`

В настоящей книге использованы главным образом объекты из двух объектных моделей. Объектная модель Access включает в себя следующие специфические только для этого приложения объекты: `Forms` (Формы), `Controls` (Элементы управления), `Reports` (Отчеты) и другие. Объектная модель ADO содержит объекты, работающие с данными. С этими объектными моделями вы познакомитесь ближе во второй и третьей части книги.

Обозреватель объектов

Как только вы начнете работать с объектами, вы увидите, что существуют сотни доступных объектов, их свойств и методов. Чтобы не заблудиться в этом дремучем лесу, воспользуйтесь услугами обозревателя объектов.

Для того чтобы открыть обозреватель объектов, выберите в меню команду `View⇒Object Browser` или нажмите клавишу <F2> (рис. 8.6).

Вот примерный порядок работы в этом окне.

- В списке `Project/Library` (Проект/Библиотека) выберите библиотеку или объектную модель, с которой будете работать. Можете также остановиться на пункте <All libraries> и посмотреть на все, что в данный момент загружено.
- Выберите класс на левой панели обозревателя. Можете также ввести часть имени нужного класса в поле поиска `Search Text` в левой верхней части окна, и таким образом ускорить работу.
- После того как класс выбран, на правой панели окна будут отображены все его свойства, события и методы (совместно их называют *членами класса*). (События рассматриваются далее в этой главе.)
- После того как выбран нужный член класса, в окне `Definition` будет отображено его краткое описание.



Рис. 8.6. Окно обозревателя объектов

- Если вам нужна более подробная справка по выбранному члену класса, щелкните на кнопке справки в верхней части окна обозревателя.
- С помощью кнопок навигации можно перемещаться вперед и назад по списку только что просмотренных членов классов.
- Для того чтобы скопировать текущий член класса в буфер обмена, щелкните на кнопке Copy to Clipboard.

Совет

При написании программ, использующих эти объекты, вы убедитесь, что применение обозревателя объектов представляет собой гораздо более быстрый способ получения справки по сравнению с поиском информации по тематическому дереву справки редактора VBE.

Создание собственных объектов

В дополнение к использованию объектов из существующих библиотек, язык VBA позволяет создавать собственные классы и объекты. В этом разделе будет показан пример создания класса, обобщающего недельный график работы. Для начала выберите в меню пункт Insert⇒Class Module (Вставить⇒Модуль класса). Открывшееся окно модуля класса будет выглядеть практически так же, как и окно обычного модуля; в то же время в окне свойств вы увидите отличающийся список параметров. Задайте в окне свойств имя нового модуля — TimeWeek.

Теперь введите текст описания создаваемого класса:


```
Option Compare Database
Option Explicit
```

```
' Описываем несколько простых свойств
Public MondayHours As Integer
Public TuesdayHours As Integer
Public WednesdayHours As Integer
Public ThursdayHours As Integer
Public FridayHours As Integer
Public SaturdayHours As Integer
Public SundayHours As Integer
```

```
' Описываем свойство, возвращаемое частной переменной
Private strEmployeeName As String
Public Property Let EmployeeName(NewName As String)
    strEmployeeName = NewName
End Property
Public Property Get EmployeeName() As String
    EmployeeName = strEmployeeName
End Property
```

```
' Вычисление трех свойств, доступных только для чтения
Public Property Get TotalHours() As Integer
    TotalHours = MondayHours + TuesdayHours + _
        WednesdayHours + ThursdayHours + FridayHours + _
        SaturdayHours + SundayHours
End Property
```

```
Public Property Get RegularHours() As Integer
    If TotalHours < 40 Then
        RegularHours = TotalHours
    Else
        RegularHours = 40
    End If
End Property
```

```
Public Property Get OvertimeHours() As Integer
    If TotalHours > 40 Then
        OvertimeHours = TotalHours - 40
    Else
        OvertimeHours = 0
    End If
End Property
```

```
' Описываем простой метод
Public Function PrintTimeReport()
    Debug.Print "Понедельник "& MondayHours
    Debug.Print "Вторник "& TuesdayHours
    Debug.Print "Среда "& WednesdayHours
    Debug.Print "Четверг "& ThursdayHours
    Debug.Print "Пятница "& FridayHours
    Debug.Print "Суббота "& SaturdayHours
    Debug.Print "Воскресенье "& SundayHours
End Function
```

В этом тексте программы продемонстрировано два способа описания свойств и одно описание метода. Простейшим способом создания свойства в

классе является обычное объявление общей переменной — все такие переменные видимы как свойства класса.

Более сложным способом создания свойства является написание *процедуры свойства*. Свойство `EmployeeName` вставлено в класс с помощью процедур `Property Let` и `Property Get`. Когда некто присваивает свойству значение, вызывается процедура `Property Let`, которой это значение передается. Когда пользователь пытается прочесть значение свойства, вызывается процедура `Property Get`, которая его и возвращает. В нашем примере значение свойства `EmployeeName` хранится во внутренней частной переменной; в данном случае процедура свойства передает это значение общей переменной.

Остальные три свойства (`TotalHours`, `RegularHours` и `OvertimeHours`) внедрены с помощью процедур `Property Get` — соответствующих процедур `Property Let` для них не создавалось. Таким образом, эти свойства доступны только для чтения, так как средств для присвоения им значения извне просто не существует.

Процедура `PrintTimeReport` является методом класса. Любая объявленная в модуле класса общая функция или процедура автоматически становится методом класса.

После сохранения этой программы вы можете использовать класс `TimeWeek` в своих целях. Вот пример вызова этого класса:

```
Sub UseTimeWeek()
    ' Пример использования класса TimeWeek
    Dim tw As New TimeWeek
    tw.MondayHours = 8
    tw.TuesdayHours = 9
    tw.WednesdayHours = 8.5
    tw.ThursdayHours = 8
    tw.FridayHours = 7.5
    tw.SaturdayHours = 4
    tw.SundayHours = 3.5
    Debug.Print "Рабочее время: "& tw.RegularHours
    Debug.Print "Сверхурочные: "& tw.OvertimeHours
    Debug.Print "Итого: "& tw.TotalHours
    Debug.Print "-----"
    tw.PrintTimeReport
End Sub
```

А вот результат вызова процедуры `UseTimeWeek` в окне `Immediate`:

```
Рабочее время: 40
Сверхурочные: 9
Итого: 49
```

```
-----
Понедельник 8
Вторник 9
Среда 8
Четверг 8
Пятница 8
Суббота 4
Воскресенье 4
```


Работа с событиями

Существует еще один аспект работы с объектами, о котором мы еще не говорили, — это *события*. Вы, вероятно, уже знакомы с событиями из опыта работы с интерфейсом пользователя Access. События являются своего рода сигналами, которые система генерирует при возникновении определенных ситуаций. В частности, с командными кнопками связаны события Click, которые возникают при щелчке на кнопке пользователем. В качестве реакции на событие можно запрограммировать какой-либо сценарий действий приложения (макрос).

Однако оказывается, что запускаемые сценарии реакции на события представляют собой горячие клавиши, введенные разработчиками Access для того, чтобы сделать возможным написание достаточно сложных приложений практически без написания собственных программ. Большинство профессиональных программистов избегают использования макросов, и на то есть определенные причины (в частности то, что в макросах невозможен перехват событий). Вместо использования макросов в качестве реакции на события можно написать специальные фрагменты кода VBA, называемые *процедурами обработки событий*.

В базе данных примеров TimeTrack во всех кнопках задействованы макросы. Преобразуем один из этих сценариев в процедуру обработки события, чтобы на примере объяснить ее синтаксис.

1. Откройте форму SwitchBoard (Панель переключателей) в режиме конструктора.
2. Выделите кнопку Клиенты и отобразите ее свойства.
3. Измените имя кнопки на cmdClients. При написании процедур обработки событий хорошей практикой является использование смысловых имен для элементов управления; тогда в тексте программы совершенно очевидно, о каком элементе идет речь.
4. Измените значение свойства Нажатие кнопки с OpenClients (это имя вызываемого при нажатии макроса) на [Процедура обработки событий] (это значение вместе с квадратными скобками вы найдете в выпадающем списке).
5. Щелкните на кнопке с тремя точками в строке этого свойства и откроется окно процедуры обработки этого события в редакторе VBE. (Если процедура еще не существует, откроется окно с ее макетом, состоящим из описания и завершающей строки.)
6. Заполните тело процедуры следующими инструкциями:

```
Private Sub cmdClients_Click()  
    DoCmd.OpenForm "Clients"  
End Sub
```
7. Сохраните форму и переключитесь в окно программы Access. Перейдите к форме Clients и щелкните на кнопке Клиенты (чтобы проверить

отработку только что созданной процедуры обработки события, в ее теле можно установить контрольную точку).

В языке VBA в ответ на события выполняются процедуры обработки событий, использующие в своих именах определенные соглашения и размещенные в соответствующих модулях. В данном случае процедура обработки события щелчка на кнопке находится в модуле формы Switchboard, и в качестве имени используется конструкция *имя_объекта_имя_события*. На рис. 8.7 показаны отличительные признаки, дающие понять, что вы работаете в процедуре обработки событий. Если вы хотите создать процедуру обработки события, не используя интерфейс пользователя Access, то можете выбрать соответствующий объект и событие в списках редактора VBE.

В данной процедуре обработки события использован еще один встроенный объект Access — DoCmd. Этот объект вы можете себе представить как связующий мостик между миром макросов Access и программой на языке VBA. Объект DoCmd не имеет свойств, но для каждого из действий, предусмотренных для макросов в Access, имеет соответствующий метод. Все, что можно сделать с помощью сценария Access, можно внедрить в код VBA с помощью объекта DoCmd.

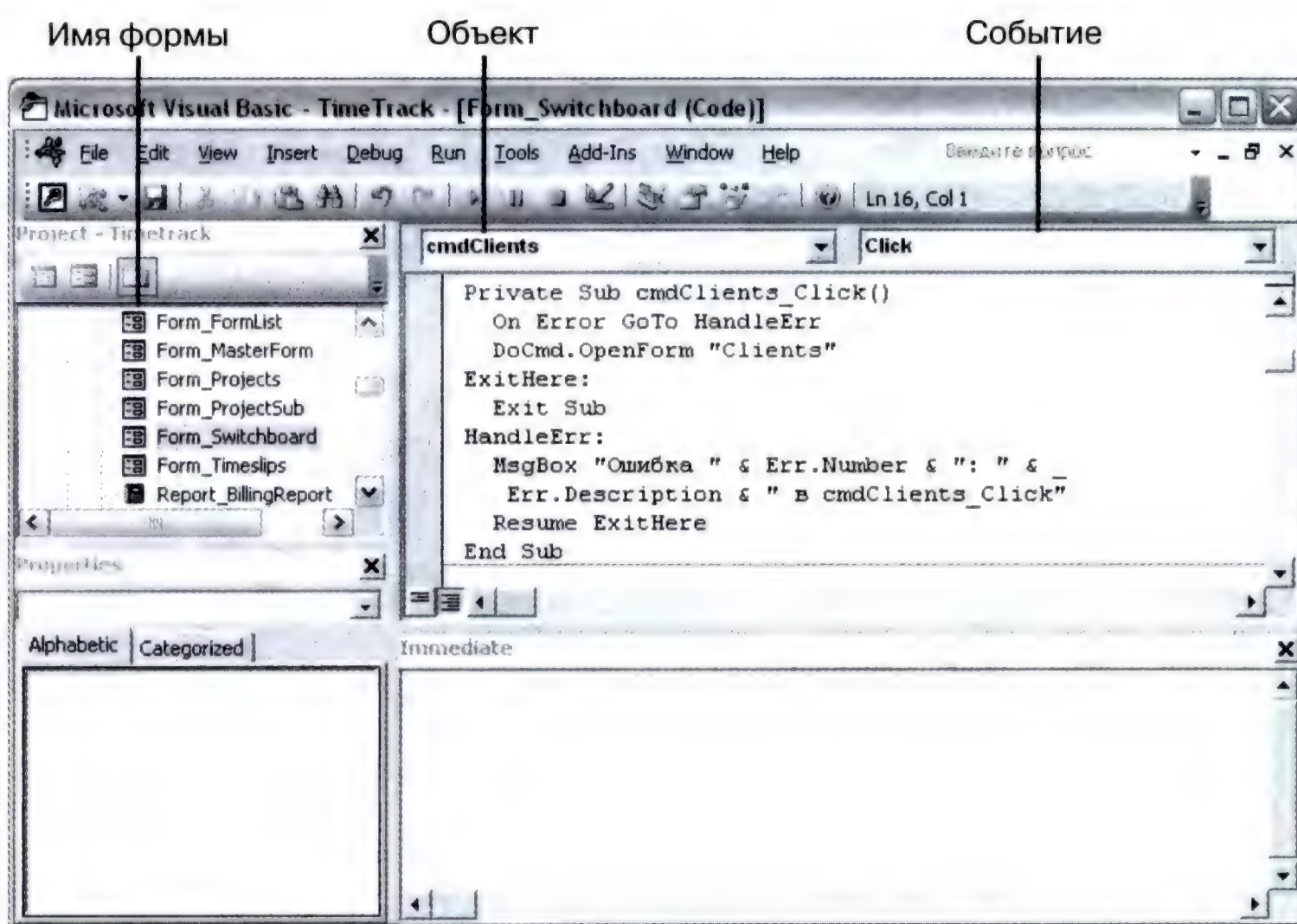


Рис. 8.7. Процедура обработки события в редакторе VBE

ПРАКТИКУМ

Открытие форм и обработка ошибок

После того как вы освоитесь с языком VBA, вам непременно захочется создавать собственные процедуры обработки событий или преобразовывать существующие макросы в процедуру событий. Это позволит впоследствии выявить ошибки, воз-

никающие при обработке событий и, к тому же, хранить всю программу в одном месте (в редакторе VBE), а не распылять ее по множеству макросов.

База данных TimeTrack содержит шесть макросов. Пять из них вызываются из формы Switchboard и служат для открытия других форм. Вот пример программы, которым можно заменить все эти макросы и вставить собственную обработку ошибок:

```
Option Compare Database
Option Explicit

Private Sub cmdBillingReport13_Click()
    On Error GoTo HandleErr
    DoCmd.OpenForm "BillingReportSetup13"
ExitHere:
    Exit Sub
HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description & "в cmdBillingReport13_Click"
    Resume ExitHere
End Sub

Private Sub cmdClients_Click()
    On Error GoTo HandleErr
    DoCmd.OpenForm "Clients"
ExitHere:
    Exit Sub
HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description & "в cmdClients_Click"
    Resume ExitHere
End Sub

Private Sub cmdEmployees_Click()
    On Error GoTo HandleErr
    DoCmd.OpenForm "Employees"
ExitHere:
    Exit Sub
HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description & "в cmdEmployees_Click"
    Resume ExitHere
End Sub

Private Sub cmdProjects_Click()
    On Error GoTo HandleErr
    DoCmd.OpenForm "Projects"
ExitHere:
    Exit Sub
HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description
    Resume ExitHere
End Sub

Private Sub cmdTimeslips_Click()
    On Error GoTo HandleErr
```



```

DoCmd.OpenForm "Timeslips"
ExitHere:
Exit Sub
HandleErr:
MsgBox "Ошибка " & Err.Number & ": " & _
    Err.Description
Resume ExitHere
End Sub

```

Предупреждение

Если вы будете вводить эту программу самостоятельно, помните, что вам нужно изменить и названия кнопок формы так, чтобы они соответствовали программе.

В этом фрагменте программы продемонстрирована простая модель обработки ошибок, обычно применяемая по умолчанию, когда не требуется производить более сложные действия. Каждая из процедур начинается с инструкции `On Error GoTo`, за ней следует метка, к которой будет осуществляться безусловный переход при возникновении ошибок. Завершается выполнение каждой процедуры оператором `Exit Sub`, возвращающим управление форме. Именно этот оператор выполняется первым, если при открытии формы не возникает ошибок; ему же передается управление после обработки ошибок в случае наличия таковых с помощью оператора `Resume`. Внутри тела фрагмента обработки ошибки открывается окно сообщения, отображающее некоторые свойства встроенного объекта `Err`. В этом объекте хранится информация о последней ошибке.

➔ Основы обработки ошибок изложены в разделе “Обработка ошибок” главы 4.

Последний макрос базы данных `TimeTrack` используется формой `BillingReportSetup13` для открытия отчета. Преобразование этого сценария в процедуру VBA будет лишь немного сложнее обычного открытия формы, поскольку дополнительно потребуется указать, что отчет должен открываться в режиме предварительного просмотра:

```

Private Sub cmdOpenReport_Click()
On Error GoTo HandleErr
DoCmd.OpenReport "BillingReport13", acViewPreview, , , _
    chkSummary
ExitHere:
Exit Sub
HandleErr:
MsgBox "Ошибка " & Err.Number & ": " & _
    Err.Description & "в cmdBillingReport13_Click"
Resume ExitHere
End Sub

```

Если с помощью обозревателя объектов посмотреть на метод `OpenReport` объекта `DoCmd`, то можно увидеть, что он имеет один параметр, содержащий список аргументов соответствующего макроса. Вторым параметром данного метода является способ открытия отчета. В данном случае этому аргументу мы передаем зна-

чение встроенной константы `vbViewPreview`, определяющее режим предварительного просмотра отчета. В общем случае аргументы метода `DoCmd` следуют в том же порядке, что и аргументы соответствующего макроса, что значительно упрощает процесс преобразования макросов в процедуры на языке VBA.

Область видимости и время жизни

9

Что такое область видимости

В этой главе мы заложим еще один важный кирпичик в фундамент ваших знаний о программировании. Более конкретно: мы рассмотрим вопрос, где процедура или переменная доступна в тексте программы. Две взаимно пересекающиеся концепции определяют, какие переменные может использовать процедура: *область видимости* определяет ее доступность в теле процедур и функций, а *время жизни* — наличие в ней корректных данных. Несмотря на то, что в примерах будут использованы только переменные, все сказанное применимо также и к константам.

Большинство переменных работают исключительно в теле тех процедур, в которых они объявлены. Однако бывают ситуации, когда переменную, объявленную в одной процедуре, нужно использовать и в других. Свойство доступности переменных в отдельных процедурах программы называют их *областью видимости*. Это свойство аналогично концепции, изложенной в разделе “Объявление общих и частных процедур” главы 4. Подобно тому, как процедуры могут вызываться из других модулей, к переменным можно обращаться из других процедур. Выделяют три уровня области видимости переменных:

В ЭТОЙ ГЛАВЕ

Что такое область видимости....	163
Время жизни переменных и констант.....	168
Статические переменные.....	172

- *Локальный уровень, или уровень процедуры.* Переменные существуют только в пределах процедуры, в которой объявлены.
- *Уровень модуля.* Переменные доступны всем процедурам модуля, в котором объявлены.
- *Глобальный уровень.* Переменные доступны в пределах всего приложения.

Переменные уровня процедуры

В этой книге уже было представлено множество переменных уровня процедуры, т.е. переменных, действующих только в пределах процедуры, в которой они объявлены; так что обратиться к ним можно только в ней.

Так как область видимости локальных переменных ограничена пределами только одной процедуры, в разных процедурах одного модуля могут существовать другие переменные с таким же именем. В то же время существование двух разных переменных с одинаковыми именами в пределах одной процедуры невозможно.

Для того чтобы проиллюстрировать локальный уровень видимости, откроем редактор VBE (для этого следует нажать клавиши <Alt+F11>), создадим новый стандартный модуль и введем следующую процедуру (можно использовать процедуры из модуля Chapter9 базы данных примеров TimeTrack):

```
Function ProcedureLevel1()
    dim strScope as String
    strScope = "Переменная уровня процедуры"
    Debug.Print strScope
End Function
```

```
Function ProcedureLevel2()
    strScope = "Здесь нет объявления переменной уровня процедуры"
    Debug.Print strScope
End Function
```

Примечание

Модуль главы 9 (Chapter9) в базе данных примеров содержит эти процедуры в несколько видоизмененном виде, так как они отражают все изменения, произошедшие с модулем в течение всей главы. Если вы хотите начать работу с этим модулем с нуля, лучше создайте его заново в своей базе данных.

Установив курсор в пределах процедуры ProcedureLevel1, нажмите клавишу <F5>. В данной процедуре переменной strScope присваивается строка "Переменная уровня процедуры", после чего данное значение выводится в окно Immediate.

Обратите внимание, что в процедуре ProcedureLevel2 происходит обращение к переменной с таким же именем. Установив курсор в пределах этой процедуры, нажмите <F5>. На этот раз интерпретатор вернет сообщение об ошибке, показанное на рис. 9.1. Причиной ошибки стало то, что переменная strScope не определена в процедуре ProcedureLevel2. Щелкните на кнопке ОК, после чего щелкните на кнопке Reset панели инструментов.

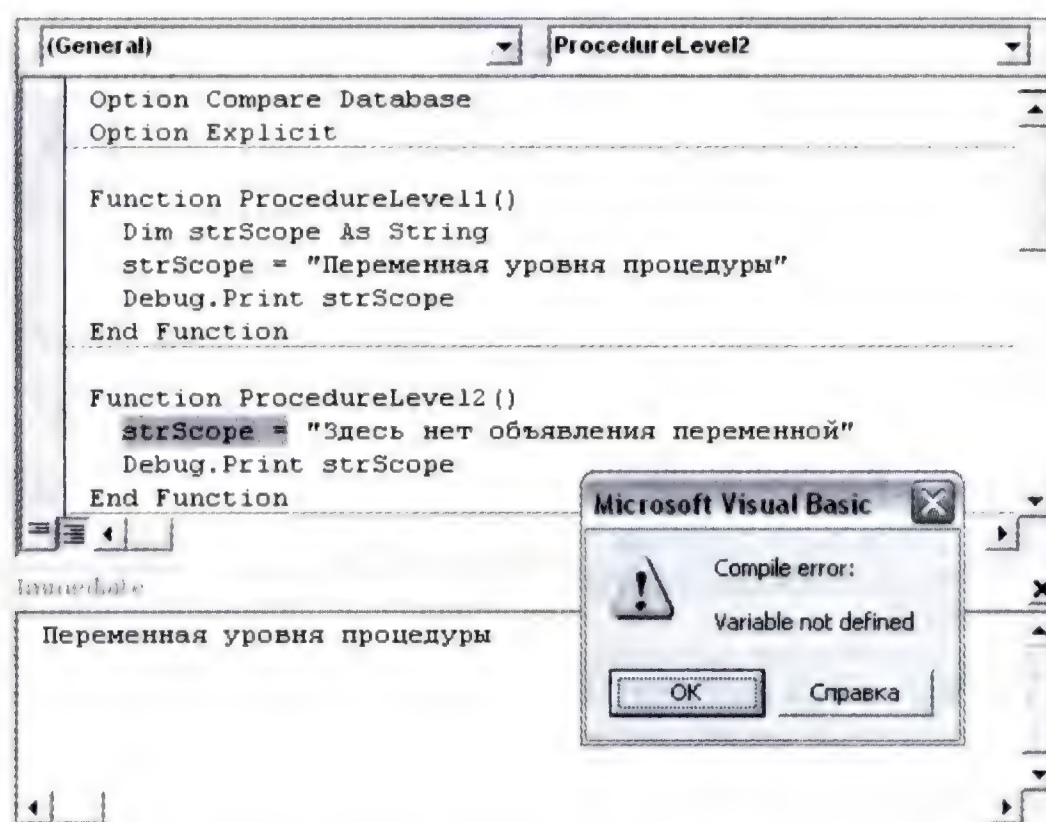


Рис. 9.1. Обращение к несуществующей переменной приводит к ошибке

Есть два способа исправления этой ошибки:

- объявление переменной `strScope` в обеих процедурах;
- объявление переменной `strScope` на уровне модуля.

Переменные и константы уровня модуля

Переменные уровня модуля доступны во всех процедурах в пределах данного модуля. Такие переменные объявляются в области общих объявлений модуля (рис. 9.2), а не в самих процедурах.

Для того чтобы проиллюстрировать данную ситуацию, переместим строку с инструкцией `Dim` из процедуры в область общих объявлений. После этой операции вызов обеих процедур не приводит к возникновению ошибок. Чтобы в этом убедиться, выполните следующие действия.

1. Используя рис. 9.2 как путеводитель, удалите инструкцию `dim strScope as String` из процедуры `ProcedureLevel1` и скопируйте ее в область общих объявлений.
2. После перемещения объявления переменной установите курсор в пределах процедуры `ProcedureLevel1` и нажмите клавишу `<F5>`. В окне `Immediate` будет выведена строка "Переменная уровня процедуры".
3. Теперь установите курсор в пределах процедуры `ProcedureLevel2` и снова нажмите клавишу `<F5>`. На этот раз выполнение процедуры пройдет без ошибок и в окне `Immediate` будет выведена строка "Здесь нет объявления переменной уровня процедуры" (рис. 9.3). Это произошло потому, что переменная `strScope` стала доступна всем процедурам модуля.

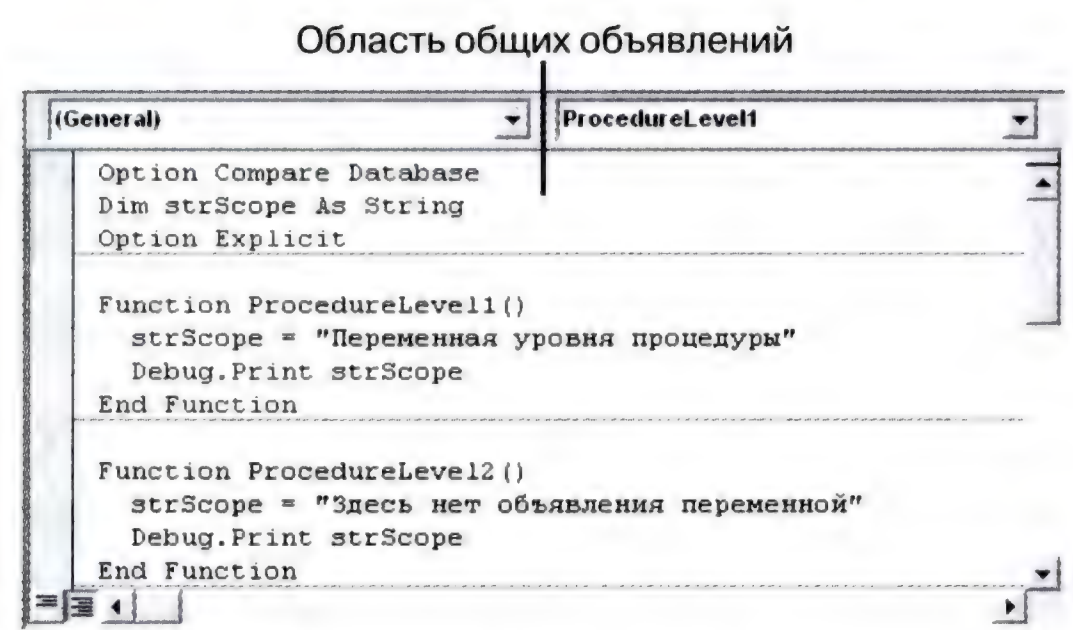


Рис. 9.2. Переместите инструкцию объявления переменной `strScope` из процедуры в область общих определений

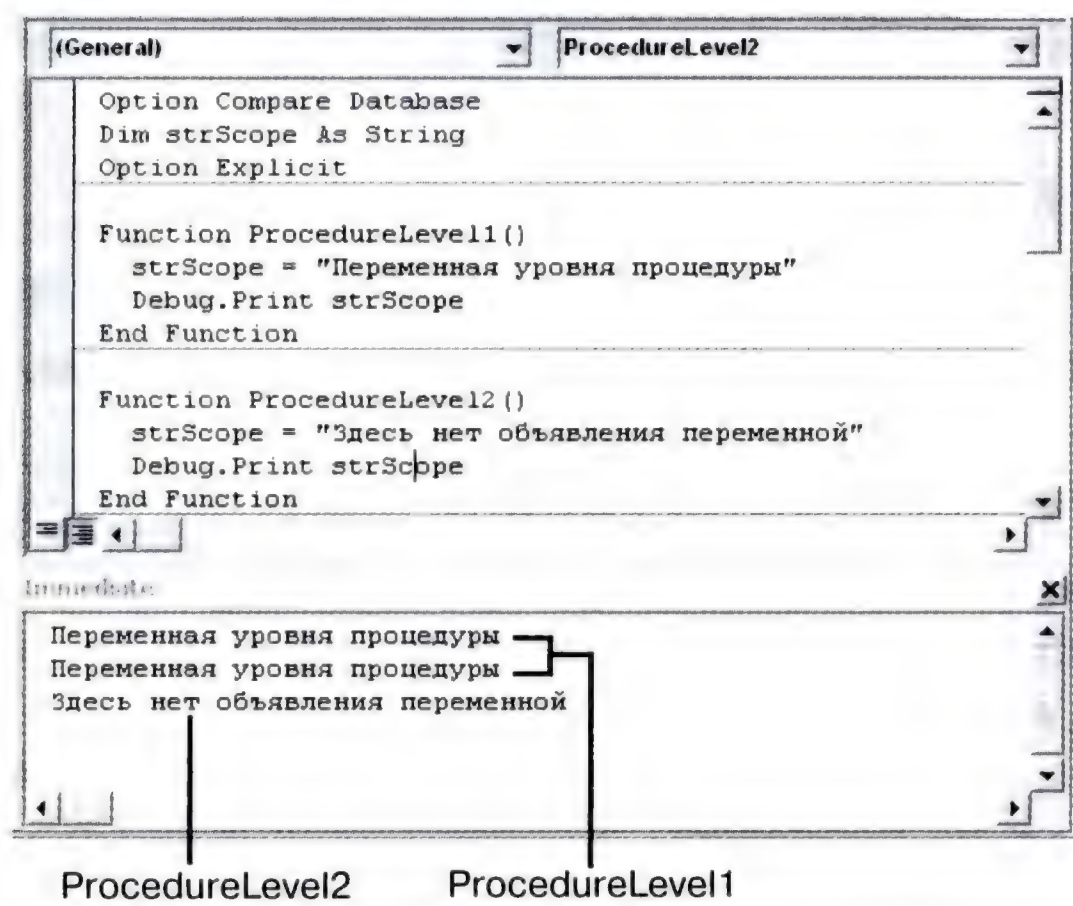


Рис. 9.3. Обе процедуры используют переменную уровня модуля, объявленную в области общих объявлений

Глобальные переменные и константы

Глобальный уровень является третьей и последней областью видимости, о которой мы будем говорить в этой главе. Попробуем протестировать переменную `strScope` на предмет ее глобальности в настоящий момент. Для этого выполните следующие действия:

1. Создайте новый стандартный модуль, выбрав в меню пункт `Insert⇒Module`.
2. Введите в пустой модуль следующую процедуру:

```
Function PublicVariableTest()
    strScope = "Теперь переменная стала общей"
    Debug.Print strScope
End Function
```


или откройте модуль `Chapter9PublicExample` в базе данных примеров `TimeTrack`.

3. Установив курсор в любое место внутри процедуры, нажмите клавишу `<F5>`. Результатом выполнения процедуры станет та же ошибка, которая произошла раньше (см. рис. 9.1). Все дело в том, что переменная `strScope` оказалась невидимой в пределах текущего модуля.
4. Щелкните на кнопке `OK`, а затем на кнопке `Reset` панели инструментов.

Исправить эту ошибку можно двумя способами:

- объявить переменную во втором модуле на уровне модуля или процедуры;
- объявить переменную `strScope` в первом модуле (`Chapter9`) как глобальную.

Для того чтобы создать глобальную переменную, вернитесь в модуль `Chapter9`, не закрывая новый модуль. В разделе общих объявлений этого модуля измените ключевое слово `Dim` на `Public` (рис. 9.4).

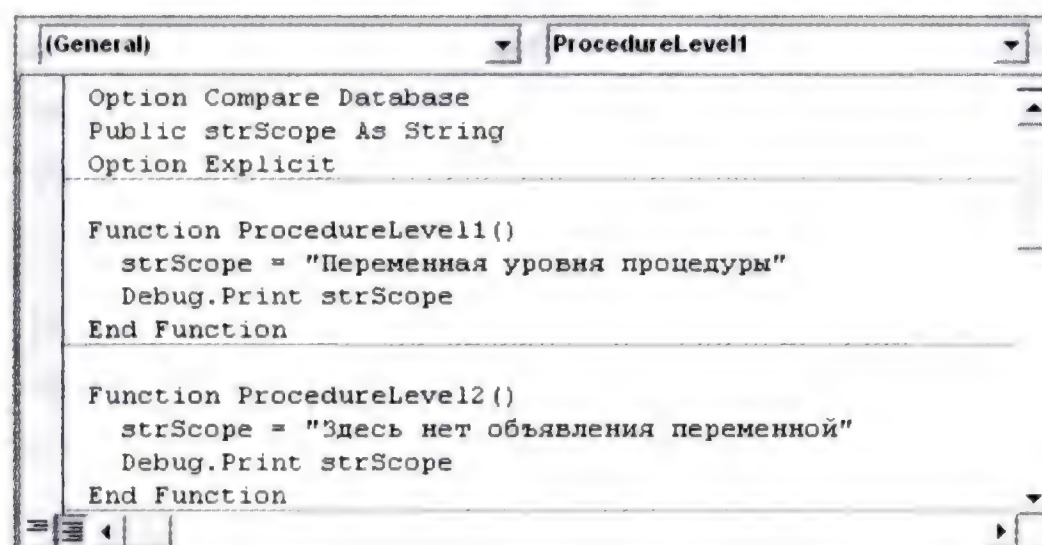


Рис. 9.4. Измените область видимости переменной с уровня модуля на глобальный

После изменения объявления переменной на общий уровень видимости вернемся во второй модуль (тот, в котором содержится процедура `PublicVariableTest`). Установив курсор в любое место этой процедуры, нажмите клавишу `<F5>`. На этот раз процедура отработает без ошибок и на печать будет выведена строка, показанная на рис. 9.5. Переменная `strScope` теперь стала общей; это значит, что ее можно использовать в любом модуле, независимо от того, в каком из них она была объявлена.

Совет

Для объявления общей константы перед ключевым словом `Const` нужно вставить ключевое слово `Public`:

```
Public Const константа As тип_данных = значение
```

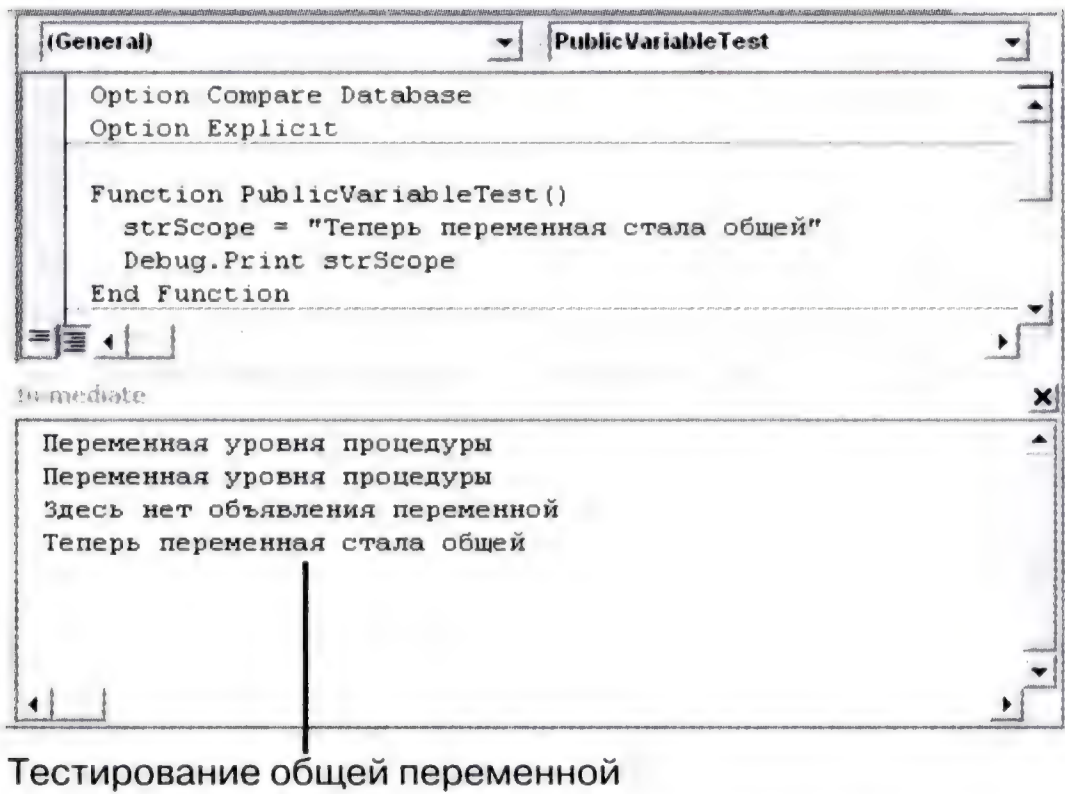



Рис. 9.5. Общая переменная доступна в любом модуле

Примечание

Модули объектов (для форм и отчетов), а также модули классов имеют дополнительный уровень области видимости, называемый *дружественным*. Дружественный уровень открывает доступ к переменной или процедуре другим объектным модулям в пределах одного проекта, подобно общему (глобальному) уровню. Отличие состоит в том, что глобальные переменные и процедуры, в отличие от дружественных процедур и переменных, доступны другим программным объектам вне базы данных Access. Для объявления дружественной переменной или процедуры вместо ключевых слов `Public` или `Private` используют ключевое слово `Friend`.

Время жизни переменных и констант

Область видимости определяет, где в тексте программы доступны переменные и процедуры. *Время жизни* же определяет, где и как долго переменная остается активной и, таким образом, содержит достоверные данные. При определении времени жизни в контексте переменной примите к сведению следующие руководства.

- Временем жизни считается время, в пределах которого переменная сохраняет свое значение.
- В течение времени жизни переменная может изменять свое значение, однако какое-то значение она все же хранит.
- Когда переменная выходит за пределы своей области видимости, она теряет свое значение. В этом прослеживается определенная взаимосвязь между временем жизни и областью видимости, однако это не одно и то же.

При выполнении процедуры интерпретатор VBA инициализирует все переменные уровня процедуры. Это значит, что всем переменным присваивается по умолчанию некоторое значение (табл. 9.1), даже если в тексте процедуры

это явным образом не указано. Установленные значения остаются в силе до тех пор, пока в некоторой инструкции переменной не будет присвоено явным образом другое значение.

Таблица 9.1. Значения инициализируемых переменных

Тип данных	Значение, присваиваемое при инициализации
Числовой	0
Строка переменной длины	" " (строка нулевой длины)
Строка фиксированной длины	Результат функции Chr (0) — непечатный символ
Тип Variant	Empty
Объект	Nothing

Теперь рассмотрим взаимосвязи между областью видимости переменной и ее временем жизни.

- Переменные уровня процедуры, объявляемые с помощью ключевого слова Dim, имеют то же время жизни, что и сама процедура. Другими словами, эти переменные сохраняют свое значение только во время работы самой процедуры.
- Переменные уровня модуля, определяемые в стандартном модуле, сохраняют свои значения до закрытия базы данных или до тех пор, пока вся программа не завершит свое выполнение. Это называют *временем жизни приложения*.
- Переменные уровня модуля, определенные в модуле класса, сохраняют свои значения до тех пор, пока открыты экземпляры объекта. Это называют *временем жизни объекта*.
- Общие (глобальные) переменные сохраняют свое значение до завершения работы программы. Общие переменные всегда имеют время жизни приложения.

Время жизни переменных уровня процедуры

Итак, время жизни переменных уровня процедуры совпадает с их областью видимости. При выполнении процедуры такие переменные инициализируются, а когда она завершается, теряют свои значения. На уровне процедуры область видимости ограничена пределами самой процедуры, а время жизни совпадает с временем выполнения самой процедуры.

В следующем примере показана переменная, изменяющая свое значение в течение времени жизни процедуры (рис. 9.6):

```
Function ProcedureLT1()  
    Dim strPLT As String  
    strPLT = "Переменная доступна"
```



```

    Debug.Print strPLT
    strPLT = strPLT & "и здесь тоже"
    Debug.Print strPLT
End Function

```

В этом примере переменная `strPLT` определяется последовательно два раза. Обратите внимание, что она не исчезает после того, как была использована. Перед каждым использованием или изменением переменной ее не нужно объявлять дополнительно. Вы можете переопределять переменную столько раз, сколько будет нужно, однако она остается доступной только во время выполнения процедуры.

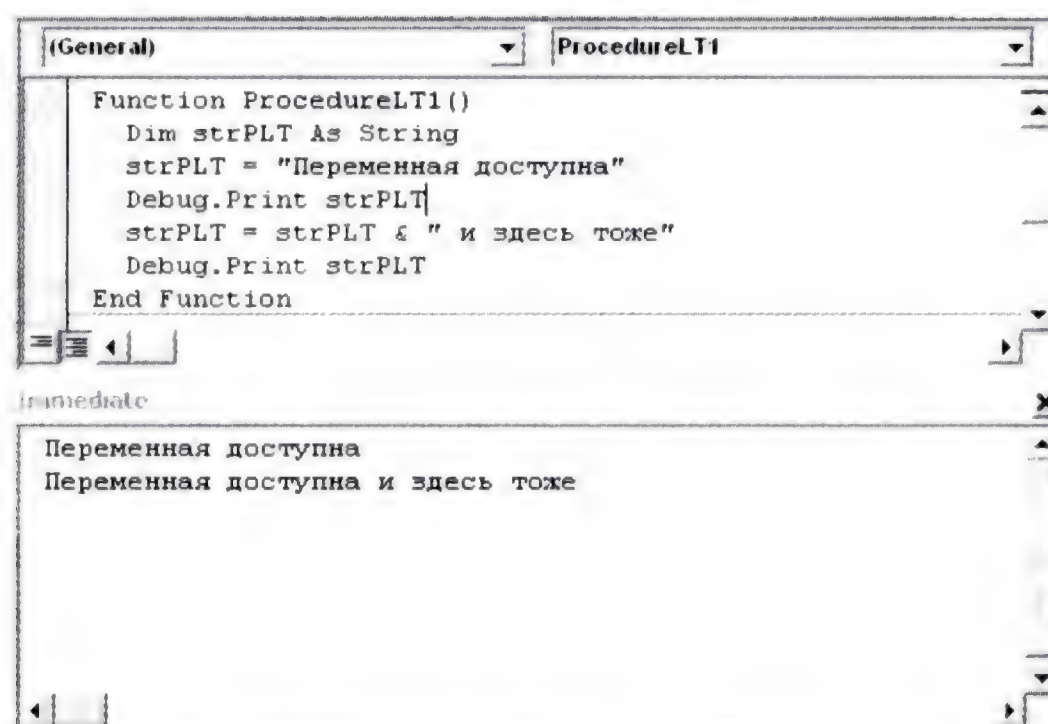


Рис. 9.6. Переменные уровня процедуры сохраняют свои значения только на время выполнения данной процедуры

Время жизни переменных уровня модуля

Теперь поднимемся на один уровень и протестируем время жизни переменных уровня модуля. При этом можно использовать существующие в модуле `Chapter9` процедуры (имеется в виду база данных примеров `TimeTrack.mdb`) следующим образом:

1. Введите в области общих определений модуля `Chapter9` следующее объявление:

```
Dim intMLT As Integer
```

2. В модуле `Chapter9` введите следующие функции:

```

Function ModuleLT1()
    Debug.Print intMLT
End Function

```

```

Function ModuleLT2()
    intMLT = 100 * 3
    Debug.Print intMLT
End Function

```

3. Установите курсор в пределах функции `ModuleLT1`, после чего нажмите клавишу `<F5>`. В окне `Immediate` будет выведено число 0, так как по-

сле инициализации и до момента присвоения реального значения переменная `intMLT` хранит именно это значение.

4. Выполните функцию `ModuleLT2`, и в окно `Immediate` будет выведено число 300, а не нуль (рис. 9.7). Переменная уровня модуля не только доступна в обеих процедурах — она сохраняет свое значение между их вызовами. Данная переменная не исчезает между отдельными вызовами процедур модуля, как это делает переменная уровня процедуры.

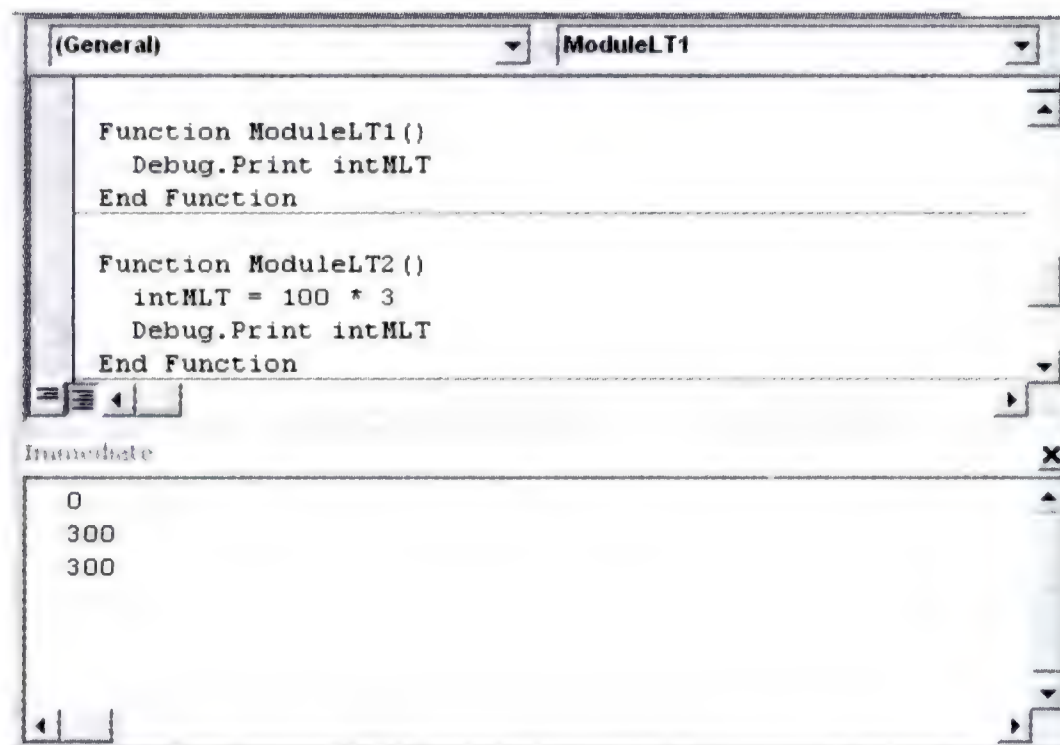


Рис. 9.7. Переменные уровня модуля хранятся в течение выполнения всей программы

Если вы попытаетесь обратиться к переменной `intMLT` в другом модуле, интерпретатор VBA ее не найдет и выдаст сообщение об ошибке выполнения. Однако при этом переменная продолжает свою жизнь в пределах своего модуля. Не путайте такое поведение с временем жизни переменной — оно объясняется ее областью видимости, а не вопросами жизни и смерти.

Время жизни общих переменных

Общие переменные доступны во всех процедурах. После того как они инициализируются, они существуют до момента своего уничтожения. В течение всего этого времени можно изменять их значения, но сама переменная текущее значение будет возвращать.

Для того чтобы проиллюстрировать долгую жизнь общей переменной, изменим объявление `intMLT` в области общих объявлений модуля `Chapter9` на `Public`. После этого мы введем в модуль `Chapter9PublicExample` следующую функцию:

```
Function PublicLTTest()  
    Debug.Print intMLT  
  
End Function
```

На рис. 9.8 показан результат выполнения функции `PublicLTTest`. Так как переменная объявлена общей, в функции `PublicLTTest` второго модуля

обращение к ней правомерно. К тому же мы видим, что значение переменной не поменялось. Переменные уровня модуля и общие переменные имеют одно и то же время жизни, но разную область видимости. Это значит, что переменная может и не быть доступной абсолютно всем процедурам приложения, но в то же время оставаться в памяти.

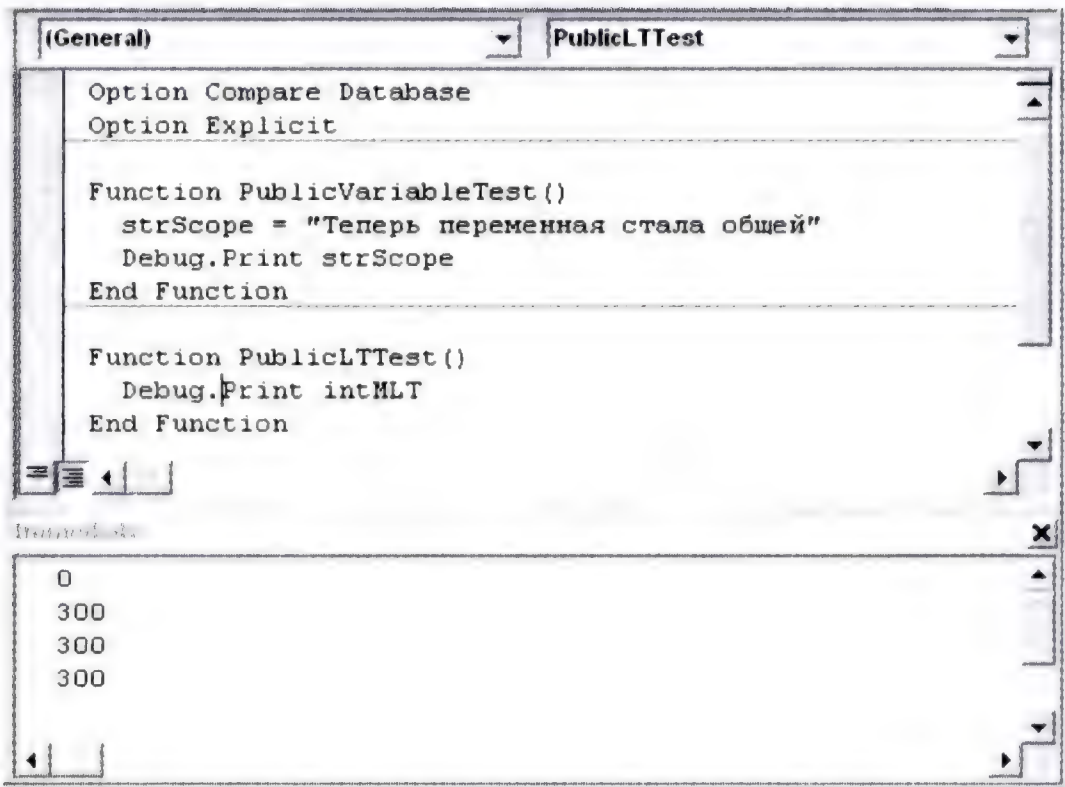


Рис. 9.8. Общие переменные остаются живыми на протяжении всего времени выполнения программы

Статические переменные

Переменные уровня процедуры нежизнеспособны вне процедуры, в которой объявлены. Однако из этого правила существует одно исключение: срок жизни таких переменных можно продлить, объявив их *статическими* с помощью ключевого слова `Static`:

```
Static имя_переменной [As тип_данных]
```

Статической называется такая переменная уровня процедуры, которая сохраняет свое значение между вызовами той процедуры, в которой она объявлена. В этом она имеет сходство с переменными уровня модуля. Если явным образом не определять `тип_данных`, статическая переменная будет иметь тип `Variant`. Таким образом, ключевое слово `Static` работает практически так же, как и `Dim`, за исключением определения времени жизни переменной.

Предупреждение

Несмотря на то, что статическая переменная сохраняет свое значение между отдельными вызовами процедуры, она невидима вне процедуры, в которой объявлена.

В следующей процедуре продемонстрирован пример использования статической переменной:

```
Function StaticVariableTest()  
    Dim intValue As Integer  
    intValue = intValue + 1
```



```

    Debug.Print intValue
End Function

```

Вы, наверное, уже догадались, какую информацию выведет в окне Immediate инструкция Debug.Print при первом обращении к ней. На рис. 9.9 показан результат первого (равно как и всех последующих) вызова функции. Процедура будет все время выводить на печать единицу, поскольку при каждом вызове переменная будет инициализироваться в нуль, после чего в следующей инструкции ее значение будет увеличиваться на единицу.

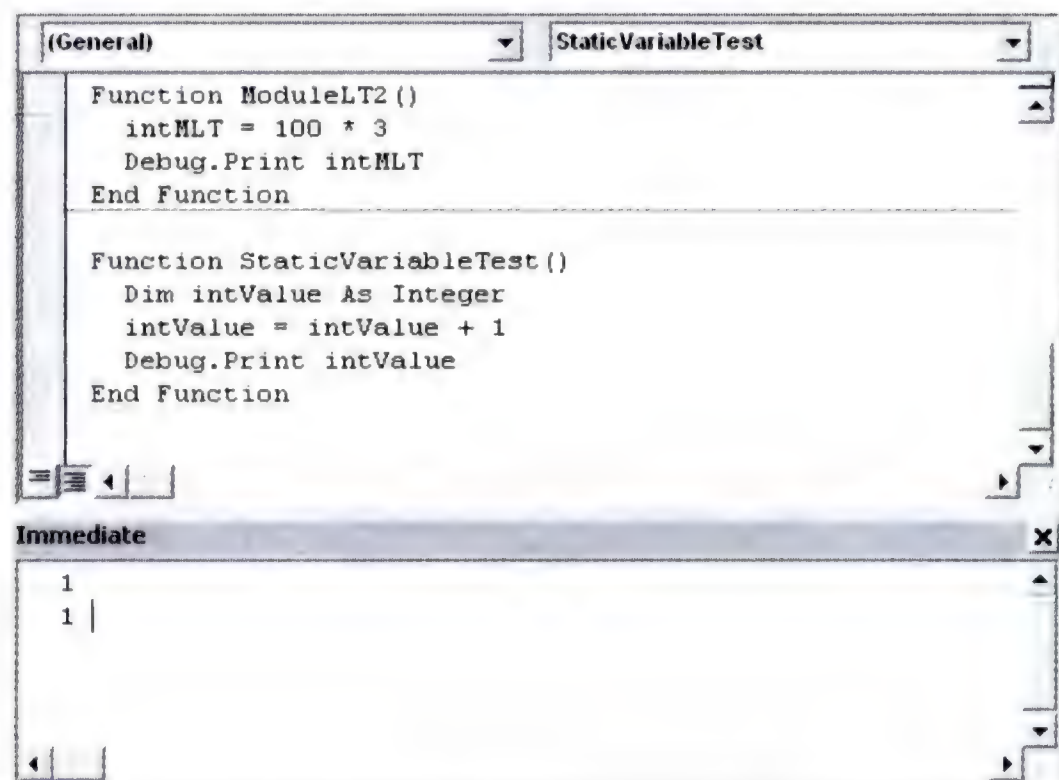


Рис. 9.9. Переменная `intValue` во время печати всегда равна единице

Если запустить эту же процедуру второй раз, на печать все равно будет выведена единица. Это происходит потому, что между отдельными вызовами переменная `intValue` теряет свое значение, и при каждом вызове она заново инициализируется. Таким образом, вам никогда не удастся получить при ее вызове значение, отличное от 1.

Теперь изменим ключевое слово `Dim` на `Static` и снова запустим процедуру. При первом запуске она, как и раньше, вернет значение 1. Запустим процедуру еще раз — теперь она вернет уже значение 2 (рис. 9.10). Это происходит потому, что теперь между отдельными вызовами переменная сохраняет свое текущее значение. Если запустить процедуру еще раз, в окно Immediate будет выведено значение 3 и т.д.

Примечание

После создания переменной объекта и присвоения ей ссылки на некоторый объект всегда включайте в программу код, который устанавливает все переменные объектов в специальное значение `Nothing` (ничто), используя следующий синтаксис:

```
Set переменная_объекта = Nothing
```

Используйте эту инструкцию для объектов, работу с которыми вы уже завершили, а также для всех остальных объектов в конце процедуры. Не забывайте включать эти инструкции во все процедуры обработки ошибок. Таким образом, если про-

цедура преждевременно прервет свою работу в состоянии ошибки, переменные объектов будут обработаны корректно. Ключевое слово `Nothing` используется только для переменных объектов и определяет, имеют ли они ссылки на реальные объекты.

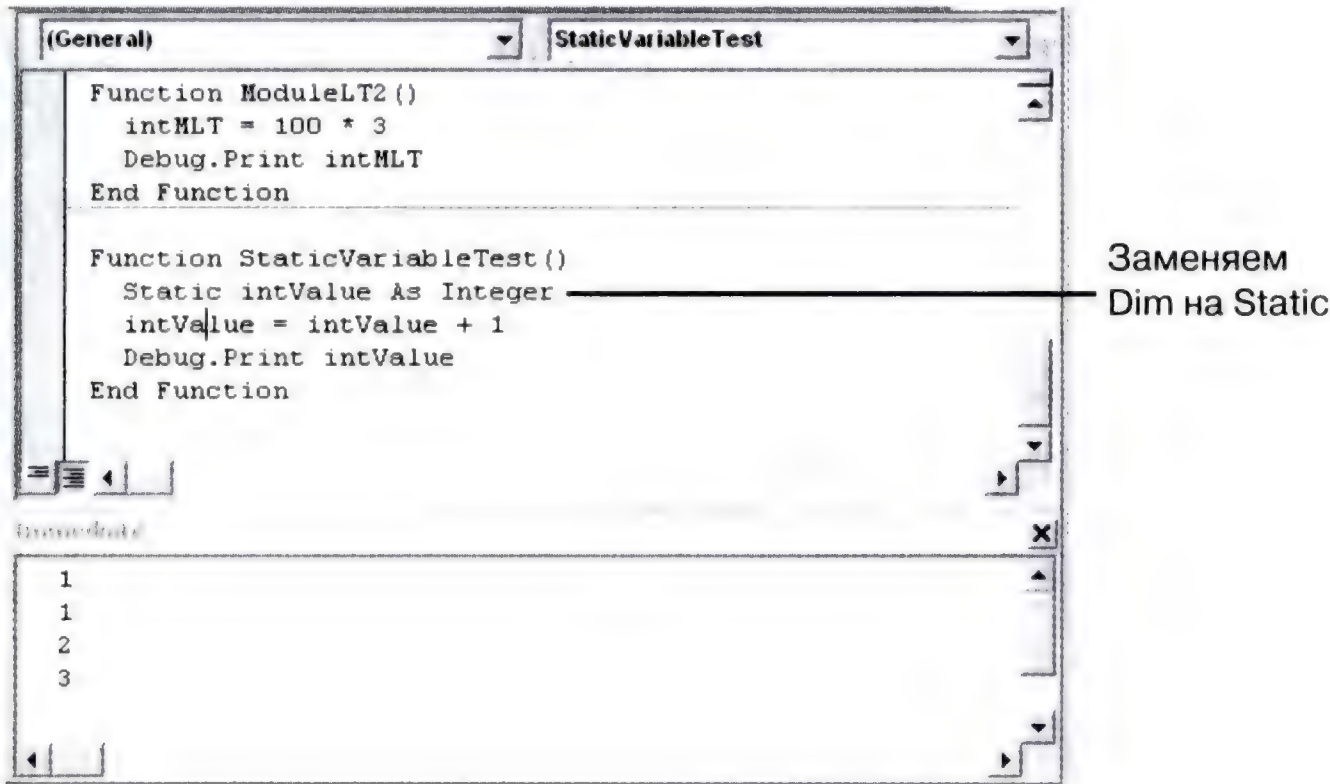


Рис. 9.10. Статическая переменная `intValue` сохраняет свое значение между вызовами процедуры

Работа с интерфейсом пользователя Access

10. Работа с формами

11. Анализ модели событий Access

12. Работа с простыми и комбинированными списками

13. Прочие элементы управления

14. Работа с отчетами

15. Работа с коллекциями

Работа с формами

10

Открытие и закрытие форм

В первой части книги вы изучали основы языка VBA. Теперь настало время применения полученных знаний в работе с Access. В этой главе мы рассмотрим вопрос использования VBA при работе с одним из самых распространенных объектов Access — формами. При создании приложений вы, несомненно, сталкивались с процессом разработки форм. Первое, что мы рассмотрим, — это процесс открытия и закрытия форм с помощью VBA.

Открытие форм

В практикуме главы 8 был представлен фрагмент программы, открывающий форму. Вернемся к этому вопросу и разберемся, что же происходит на самом деле. Объект DoCmd является ключом к множеству возможностей, автоматизации работы в Access с помощью языка VBA. Один объект DoCmd открыт все время — его не нужно ни создавать, ни уничтожать. Все, что нам потребуется — это использовать методы объекта. Эти методы позволяют программе на языке VBA взаимодействовать непосредственно с объектами Access.

Для того чтобы открыть форму, используется метод DoCmd.OpenForm, имеющий массу необязательных аргументов и следующий синтаксис:

В ЭТОЙ ГЛАВЕ

Открытие и закрытие форм.....	177
Модуль формы и обработка событий	179
Выполнение часто встречающихся задач	180
Обработка ошибок на уровне формы	188
Работа с несколькими экземплярами формы	190


```
DoCmd.OpenForm имя_формы, [вид], [фильтр], [условие], _
    [режим_данных], [режим_окна], [операнды]
```

При использовании метода `DoCmd.OpenForm` нужно обязательно указывать имя формы. Необязательные аргументы позволяют более точно управлять поведением открываемой формы.

- Аргумент *вид* определяет изначальное представление формы (форма, таблица и т.п.).
- Аргумент *фильтр* позволяет задать имя запроса, который будет использован в качестве фильтра данных, отображаемых в форме.
- Аргумент *условие* содержит выражение, используемое в качестве фильтра формы (это — аналог конструкции `WHERE` в запросе SQL, однако само ключевое слово `WHERE` в данном выражении не указывается).
- Аргумент *режим_данных* определяет изначальное устанавливаемый режим ввода данных в форме (к примеру, разрешено ли редактирование данных, или они доступны только для чтения).
- Аргумент *режим_окна* определяет, как будет открыта форма — в обычном или скрытом режиме, в виде диалогового окна или значка.
- Аргумент *операнды* позволяет передать в модуль формы дополнительные произвольные данные. Позже в этой главе мы отдельно остановимся на этом вопросе.

В своей простейшей форме метод `DoCmd.OpenForm` предполагает, что вы согласны с присвоением всем необязательным аргументам значений по умолчанию. Вот как в этом случае можно открыть форму `Clients`:

```
Sub openClientForm()
    ' Открытие формы со значениями по умолчанию
    DoCmd.OpenForm "Clients"
End Sub
```

В данном случае форма будет открыта в том представлении, в котором в последний раз была сохранена. В ней будут отображаться все данные; будет разрешено их редактирование, а сама форма будет открыта в виде окна. В то же время, если такой режим вам не подходит, можете самостоятельно установить отдельные аспекты открытия формы. К примеру, для открытия формы `Clients` в табличном представлении (независимо от того, в каком представлении эта форма была сохранена) можно создать следующую процедуру:

```
Sub OpenClientFormDataSheet()
    ' Открытие формы в табличном представлении
    DoCmd.OpenForm "Clients", View:=acFormDS
End Sub
```

Заккрытие формы

После того как форма будет открыта, для ее закрытия потребуется выполнение метода `DoCmd.Close`. Этот метод применим ко всем объектам Access, а не только к формам. В общем виде он имеет следующий синтаксис:

DoCmd.Close [тип_объекта], [имя_объекта], [сохранение]

- Аргумент *тип_объекта* определяет, какой объект вы собираетесь закрывать: форму, отчет или что-либо другое.
- Аргумент *имя_объекта* определяет имя закрываемого объекта.
- Аргумент *сохранение* определяет, следует ли сохранять изменения в объекте при его закрытии.

Если форма `Clients` уже открыта, следующая процедура ее закроет:

```
Sub CloseClientForm()
    ' Закрытие формы Clients
    DoCmd.Close acForm, "Clients"
End Sub
```

При закрытии формы система Access по умолчанию спрашивает у пользователя, следует ли сохранять изменения, произведенные с самой формой (например, новый установленный фильтр, или изменение порядка записей). Такое поведение можно изменить, задав в качестве последнего аргумента константы `acSaveNo` (отмена произведенных изменений) или `acSaveYes` (автоматическое сохранение изменений). Применим такую модернизацию к нашему примеру закрытия формы `Clients`:

```
Sub CloseClientForm()
    ' Закрытие формы Clients
    ' с автоматическим сохранением изменений
    DoCmd.Close acForm, "Clients", _
        Save:=acSaveYes
End Sub
```

Совет

Вы, наверное, уже заметили, что все аргументы метода `Close` являются не обязательными. А что произойдет, если не указать тип и имя объекта? В результате такого действия будет закрыт тот объект, который находится в момент вызова метода в фокусе.

Модуль формы и обработка событий

Все процедуры, которые вы видели до сих пор в этой главе, находились в стандартных модулях. Однако, как вы уже видели в этой книге, фрагменты программ могут быть ассоциированы с конкретными формами. Любая форма в Access может иметь свой собственный *модуль формы*. Этот модуль сохраняется как часть самой формы; так что если форма копируется в некоторую другую базу данных, вместе с ней копируется и ее программный модуль.

Одним из типов процедур, которые обычно хранят в модуле формы, являются процедуры обработки событий. Эти процедуры вызываются, если что-то происходит с формой или каким-либо из ее элементов. К примеру, щелчок на кнопке формы вызывает процедуру обработки события, ассоциированную событием `Click` данной кнопки. Чтобы продемонстрировать этот

процесс, добавим одну кнопку на форму `Clients`. Для этого выполните следующие действия:

1. Откройте форму `Clients` в режиме конструктора.
2. Добавьте на форму командную кнопку, назовите ее именем `cmdClose` и установите ее заголовок в значение `Заккрыть форму`.
3. Установите свойство `Нажатие кнопки` в значение `[Процедура обработки событий]`, после чего щелкните на кнопке с тремя точками в этой строке. В окне редактора VBE откроется модуль данной формы.
4. Введите в окне модуля формы следующую процедуру:

```
Private Sub cmdClose_Click()  
    DoCmd.Close  
End Sub
```
5. Сохраните форму и протестируйте работу новой кнопки.

Выполнение часто встречающихся задач

В сочетании с Access средства языка VBA являются довольно гибкими. В одной главе практически невозможно описать все, что можно сделать с помощью VBA. В этом разделе представлены основные методики работы с формами в коде VBA:

- проверка существования формы;
- проверка факта открытия формы;
- перемещение формы и изменение ее размеров;
- передача аргументов в форму;
- наполнение форм содержанием.

Проверка существования формы

Иногда может потребоваться узнать, существует ли в базе данных конкретная форма. Эта проверка оказывается особенно полезной, когда пишется программа общего назначения, которая может работать с различными базами данных. Программа Access поможет решить эту проблему — она имеет встроенные средства проверки существования формы. На рис. 10.1 показан фрагмент объектной модели Access, который может пригодиться в данной ситуации.

Объект с именем `CurrentProject` (Текущий проект) реализует интерфейс пользователя к объектам текущей базы данных и содержит коллекции всех основных классов объектов. Каждая из этих коллекций содержит объекты класса `AccessObject` (Объект Access).

К примеру, коллекция `AllForms` содержит по одному объекту `AccessObject` для каждой из форм в текущей базе данных. Сам объект `AccessForm` не является формой — он всего лишь указывает на наличие такой формы.

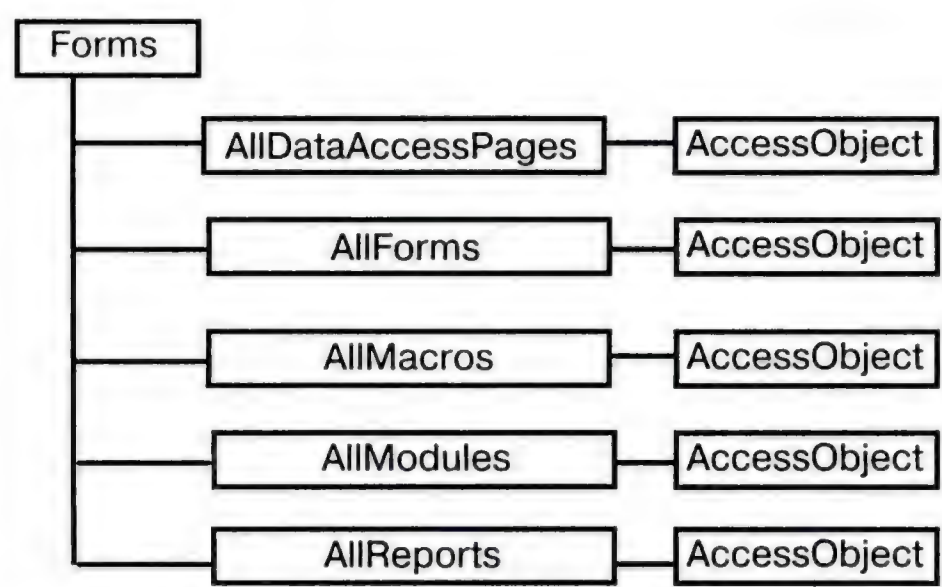


Рис. 10.1. Фрагмент объектной модели Access

Объединяя коллекцию AllForms с обработкой ошибок, можно узнать, существует ли заданная форма в текущей базе данных:

```
Function DoesFormExist(strName As String) As Boolean
    ' Проверка существования формы
    On Error GoTo HandleErr
    Dim AO As AccessObject
    Set AO = CurrentProject.AllForms(strName)
    DoesFormExist = True
ExitHere:
    Exit Function
HandleErr:
    DoesFormExist = False
    Exit Function
End Function
```

Процедура начинается с установки обработчика ошибок, передающего инструкции, следующей за меткой HandleErr при возникновении какой-либо ошибки выполнения. После этого производится попытка создания объекта AccessObject, соответствующего имени формы, которое передано пользователем в данную процедуру. Если эта попытка завершается успешно (т.е. такая форма существует), возвращаемое функцией значение устанавливается в True. Если попытка завершается провалом, возникает событие ошибки, после чего управление передается обработчику ошибок, в котором переменной процедуры присваивается значение False. В любом случае, сразу после присвоения переменной процедуры значения следует инструкция Exit Function, возвращающая управление вызывающей процедуре.

На рис. 10.2 показан вызов этой процедуры из окна Immediate.

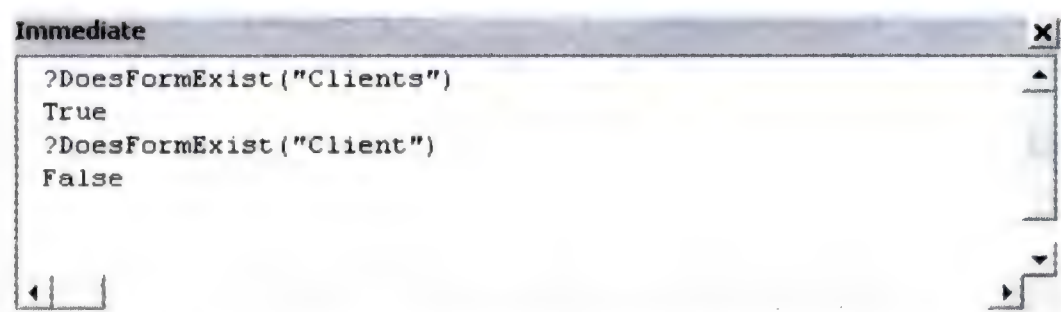


Рис. 10.2. Проверка существования формы

Примечание

Иногда функции, подобные `DoesFormExists`, приводят начинающих программистов в замешательство. У них возникает вопрос: а не вредно ли целенаправленно вызывать ошибку в тексте программы, чтобы потом использовать ее обработчик? Ответ на этот вопрос однозначен — не вредно. Если ошибка вызывается с целью сбора информации или выполнения специального фрагмента кода, такая методика является допустимой, а иногда даже незаменимой.

Проверка факта загрузки формы

Иногда оказывается недостаточным знание факта существования формы в базе данных, а требуется узнать, открыта ли в данный момент она на экране. В главе 8 мы познакомились с объектом коллекции форм, носящего название `Forms`. В нем содержатся ссылки на все открытые формы. Определив наличие искомой формы в этой коллекции, вы можете с уверенностью сказать, что она открыта. Вот фрагмент программы, выполняющий эту проверку:

```
Function IsFormOpen(strName As String) As Boolean
    ' Проверка, открыта ли форма на экране
    On Error GoTo HandleErr
    Dim frm As Form
    Set frm = Forms(strName)
    IsFormOpen = True
ExitHere:
    Exit Function
HandleErr:
    IsFormOpen = False
    Exit Function
End Function
```

Конструктивно эта процедура практически идентична той, в которой проверялось наличие формы в базе данных (см. предыдущий подраздел). Единственное отличие состоит в том, что анализируется другая коллекция.

Перемещение формы и изменение ее размеров

Метод `MoveSize` объекта `DoCmd` позволяет изменить положение на экране, а также размеры объекта, находящегося в данный момент в фокусе. Этот метод имеет четыре необязательных аргумента.

- Аргумент `Right` (Вправо) определяет расстояние от левой границы формы до левой границы рабочей области окна Access.
- Аргумент `Down` (Вниз) определяет расстояние от верхней границы формы до верхней границы рабочей области окна Access.
- Аргумент `Width` задает новую ширину формы.
- Аргумент `Height` задает новую высоту формы.

Так как среди этих аргументов содержатся высота и ширина формы, можно с помощью метода `MoveSize` изменить только размеры формы, не меняя ее

положения на экране. Рассмотрим практический пример. Форма `Clients` базы данных `TimeTrack` в настоящее время отображает как информацию о клиентах, так и информацию об их проектах. Этот объем информации, как правило, избыточен. Можно предусмотреть такое поведение, когда список проектов будет открываться только после нажатия самим пользователем на некоторую командную кнопку.

1. Откройте форму `Clients` в режиме конструктора.
2. Добавьте в нее новую командную кнопку. Установите ее имя равным `cmdProjects`, а заголовок — "Показать проекты".
3. В модуль формы введите следующий фрагмент программы:

```
Private Sub Form_Load()
    ' Устанавливаем маленький размер формы по вертикали
    DoCmd.MoveSize Height:=3345
End Sub

Private Sub cmdProjects_Click()
    ' Отображаем или скрываем раздел списка проектов
    If cmdProjects.Caption = "Показать проекты" Then
        DoCmd.MoveSize Height:=6465
        cmdProjects.Caption = "Скрыть проекты"
    Else
        DoCmd.MoveSize Height:=3345
        cmdProjects.Caption = "Показать проекты"
    End If
End Sub
```

4. Закройте и снова откройте форму. Теперь она откроется в свернутом режиме, показанном на рис. 10.3. Щелкните на только что созданной кнопке, и форма расширится в режим, показанный на рис. 10.4. Обратите внимание, что при изменении размеров формы одновременно изменяется и название кнопки. Щелкните на этой кнопке еще раз, и форма возвратится в сжатый режим.

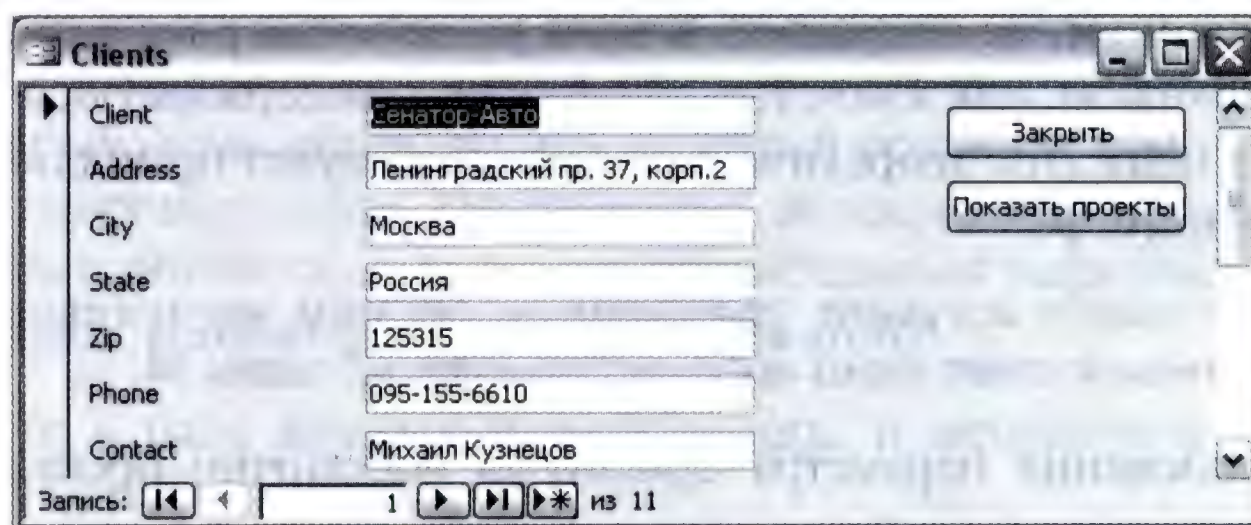


Рис. 10.3. Форма `Clients`, открытая в изначальном, сжатом режиме

В приведенном фрагменте программы на некоторые моменты стоит обратить особое внимание. Во-первых, событие `Load` формы наступает тогда, когда она открывается, а ее элементы отображаются на экране. В ответ на это

событие мы вызываем метод `MoveSize`, позволяющий установить исходные размеры формы. Во-вторых, при обработке события `Click` командной кнопки мы проверяем текущий ее заголовок, чтобы узнать, в каком режиме открыта форма в настоящий момент, и следует ее сворачивать или разворачивать. При использовании этой методики вам не потребуется создавать отдельную переменную для хранения текущего режима отображения формы.

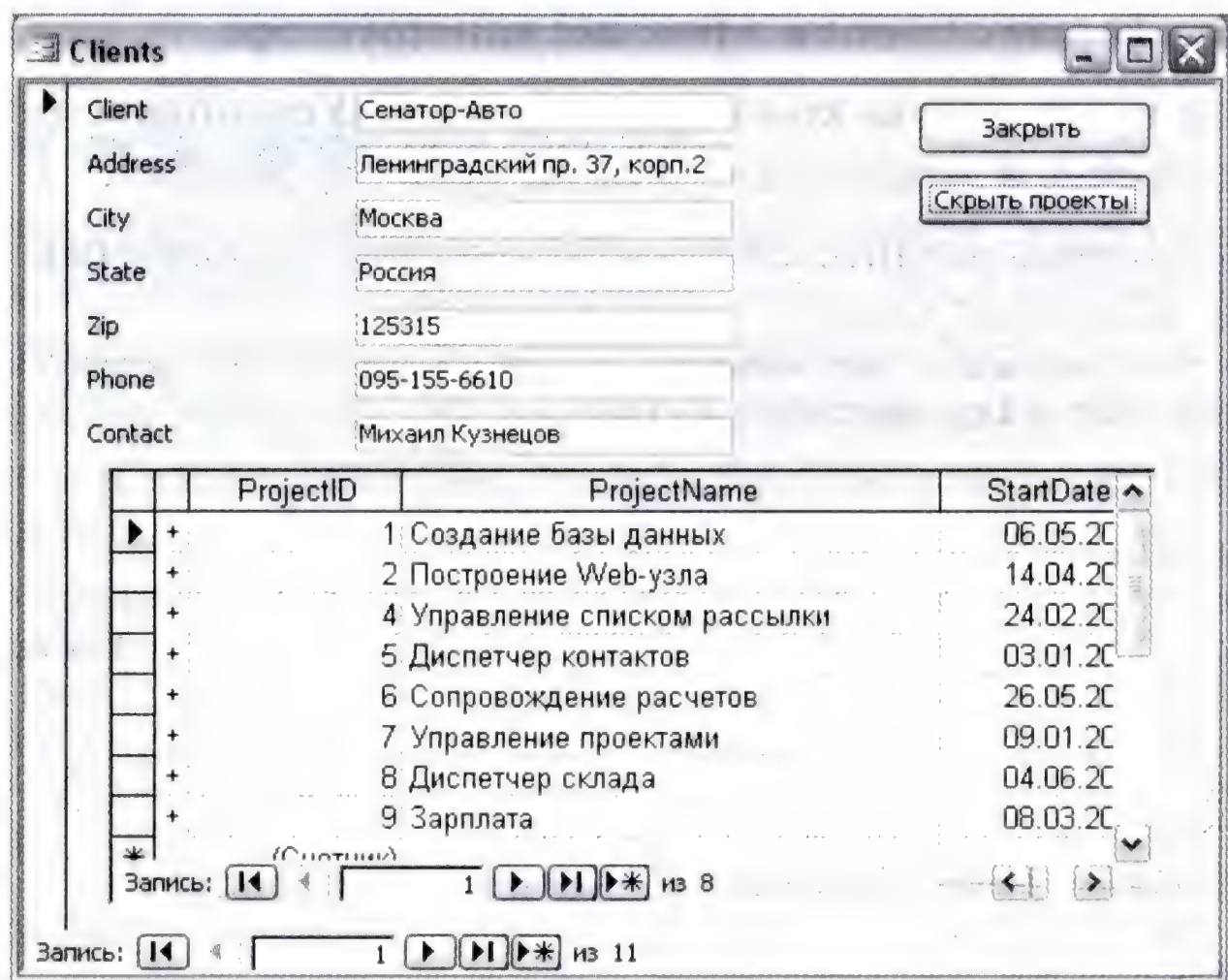


Рис. 10.4. Форма Clients, открытая в расширенном режиме

Передача данных в форму через аргумент `OpenArgs`

Параметр `OpenArgs` является одновременно как аргументом, передаваемым методу `DoCmd.OpenForm`, так и свойством самой формы. Причина такого двойного использования заключается в возможности передачи информации в форму в момент ее открытия. В этом разделе мы рассмотрим синтаксис `OpenArgs` и покажем, как извлечь информацию, содержащуюся в этой структуре. В конце главы этот теоретический материал будет проиллюстрирован на практическом примере.

➔ Отчеты также имеют аргумент `OpenArgs`, и об этом мы поговорим в разделе “Передача данных в отчет через аргумент `OpenArgs`” главы 14.

Идея использования параметра `OpenArgs` достаточно проста: с помощью этого аргумента в метод `OpenForm` можно передать произвольную строку данных, которая затем будет использована в форме. Значение этого аргумента будет доступно в любом месте модуля формы. Рассмотрим этот вопрос на примере:

1. Создайте новую форму в режиме конструктора. Поместите на эту форму элемент текстового поля и присвойте ему имя `txtOpenOrgs`.

2. Добавьте в модуль формы следующий текст:

```
Private Sub Form_Load()
    txtOpenArgs.SetFocus
    txtOpenArgs.Text = Me.OpenArgs
End Sub
```

3. Сохраните форму под именем Chapter10Test.
4. Закройте форму.
5. В окне Immediate выполните следующую инструкцию, что приведет к повторному открытию созданной формы:
DoCmd.OpenForm "Chapter10Test", OpenArgs:="Передаются эти данные"
6. Откроется форма, и в ней будет отображено значение, переданное через аргумент OpenArgs.

Совет

Для установки большинства свойств элементов управления требуется, чтобы эти элементы были в фокусе. Каждый из элементов, который может получить фокус, имеет свой метод `SetFocus`, который можно вызывать для гарантии активности данного элемента.

В дополнение к параметру `OpenArgs` в этом фрагменте программы содержится также ключевое слово `Me`. На общедоступном языке это ключевое слово значит следующее: “объект, который содержит данный программный код”. Когда программа запускается в модуле формы, ключевое слово `Me` ссылается на саму форму. Следовательно, ссылка `Me.OpenArgs` указывает на значение свойства `OpenArgs` данной формы.

Наполнение форм содержанием

До сих пор мы работали только с внешним видом форм, но никак не манипулировали ее данными. Для примера предположим, что одну и ту же форму нужно использовать для нескольких разных множеств данных. На текущий момент форма `Timeslips` отображает все затраты времени, зарегистрированные в базе данных. Это — избыточный объем данных, который усложнит поиск определенной информации, например, работ, выполненных за последнюю неделю. Попробуем скомбинировать свойство `OpenArgs` с дополнительным запросом, ограничивающим объем данных, отображаемых в форме. Для этого:

1. Создайте в базе данных новый запрос и добавьте в него таблицу `Timeslips`.
2. Переключитесь в представление SQL и введите следующий запрос на языке SQL:

```
SELECT Timeslips.*
FROM Timeslips
WHERE (Timeslips.DateWorked Between Now() - 7 And Now())
ORDER BY Timeslips.DateWorked
```


3. Сохраните этот запрос под именем `qryWeeklyTimeslips`.
4. Откройте форму `Timeslips` в режиме конструктора. Переключитесь в модуль формы и введите в него следующий фрагмент:

```
Private Sub Form_Open(Cancel As Integer)
    ' Установить источник данных в зависимости от того,
    ' как вызвана форма
    Select Case Me.OpenArgs
        Case "All"
            Me.RecordSource = "SELECT Timeslips.* "&
                "FROM Timeslips ORDER BY Timeslips.DateWorked;"
        Case "Week"
            Me.RecordSource = "WeeklyTimeslips"
            cmdWeek.Visible = False
        Case Else
            Me.RecordSource = "SELECT Timeslips.* "&
                "FROM Timeslips ORDER BY Timeslips.DateWorked;"
    End Select
End Sub
```

5. Сохраните форму. Для тестирования этой программы введите соответствующую команду `OpenForm` в окне `Immediate`. К примеру, первая из нижеприведенных строк открывает форму с отображением всех записей таблицы, а вторая — с отображением данных только за последнюю неделю:

```
DoCmd.OpenForm "Timeslips", OpenArgs:="All"
DoCmd.OpenForm "Timeslips", OpenArgs:="Week"
```

Естественно, все эти команды можно вызвать и из интерфейса пользователя. К примеру, можно создать новую кнопку на форме `Switchboard`, открывающую сводку за неделю. Таким образом, используя одну и ту же форму с различными наборами данных, можно сделать приложение более легким в сопровождении.

Совет

В большинстве случаев оператор `Case Else` используют, закладывая в него поведение, принятое по умолчанию. В данном примере, если пользователь передает недопустимое значение аргумента, форма отобразит все записи таблицы.

- ➔ Обращение к источнику данных в свойстве `RecordSource` происходит не в процедуре обработки события `Load` (Загрузка), а в процедуре обработки события `Open` (Открытие). Причина заключается в том, что во время открытия формы менять источник данных уже поздно. Более подробно порядок возникновения событий рассматривается в соответствующем разделе главы 11.

Ключом к данным, отображаемым в форме, является свойство `RecordSource` формы. Значением этого свойства может быть выражение на языке SQL или просто имя запроса, как в описанном выше примере. Если вы не хотите создавать запрос, а планируете просто отображать все записи некоторой таблицы, этому свойству можете присвоить имя таблицы.

В качестве альтернативы установке свойства `RecordSource` вы можете уже после открытия формы установить фильтр. Для этого используют метод `DoCmd.ApplyFilter`. Усовершенствуем форму `Timeslips`.

1. Откройте форму `Timeslips` в режиме конструктора.
2. Добавьте в форму новую кнопку. Установите для этой кнопки заголовок "Показать неделю" и имя `cmdWeek`.

3. Измените модуль формы следующим образом:

```
Private Sub cmdweek_Click()
    'Установка фильтра на последнюю неделю
    DoCmd.ApplyFilter "WeeklyTimeslips"
End Sub

Private Sub Form_Open(Cancel As Integer)
    ' Установить источник данных в зависимости от того,
    ' как вызвана форма
    Select Case Me.OpenArgs
        Case "All"
            Me.RecordSource = "SELECT Timeslips.* "& _
                "FROM Timeslips ORDER BY Timeslips.DateWorked;"
        Case "Week"
            Me.RecordSource = "WeeklyTimeslips"
            cmdWeek.Visible = False
        Case Else
            Me.RecordSource = "SELECT Timeslips.* "& _
                "FROM Timeslips ORDER BY Timeslips.DateWorked;"
    End Select
End Sub
```

4. Сохраните и откройте форму, после чего щелкните на только что созданной кнопке.

Примечание

Метод `ApplyFilter` объекта `DoCmd` позволяет применить фильтр, выполнить запрос или задать условие SQL `WHERE` к таблице, форме или отчету с целью ограничения или сортировки данных. Этот метод имеет следующий синтаксис:

```
DoCmd.ApplyFilter [имя_фильтра] [, условие]
```

Оба аргумента метода являются не обязательными, однако нужно задать хотя бы один из них. В случае, когда указано оба аргумента, применяется аргумент `условие`.

Обратите внимание на модификацию процедуры `Form_Open` для случая, когда источник данных уже отфильтрован — здесь нет никакого смысла отображать кнопку на форме, поскольку она ни на что не повлияет.

- ➔ В примерах для фильтрации записей было использовано выражение SQL `SELECT`. Более подробно синтаксис этого выражения описан в Приложении в конце книги.

Обработка ошибок на уровне формы

В формах Access пользователь взаимодействует напрямую с механизмом *Jet*, и именно здесь наиболее вероятно возникновение ошибок. Во-первых, пользователь может ввести данные, которые выходят за пределы допустимых, попытаться создать запись, дублирующую первичный ключ, или установить циклическую ссылку.

Примечание

Jet — основной механизм работы с базой данных в Access. Это составная часть приложения, отвечающая за сохранение и извлечение данных.

Ни одна из этих ошибок не перехватывается в коде VBA, так как VBA не является посредником между механизмом Jet и интерфейсом пользователя. В то же время, существует другой способ собственной обработки ошибок — обработка события `Error` формы. Это событие генерируется, когда при обращении механизма Jet к данным происходит какая-либо ошибка. Вы можете изменить стандартную реакцию системы на ошибку, написав собственную процедуру ее обработки. Для этого выполните следующие действия:

1. Откройте форму `Timeslips` и попробуйте ввести новую запись. В качестве даты введите слово "Вторник" — появится сообщение об ошибке, показанное на рис. 10.5.

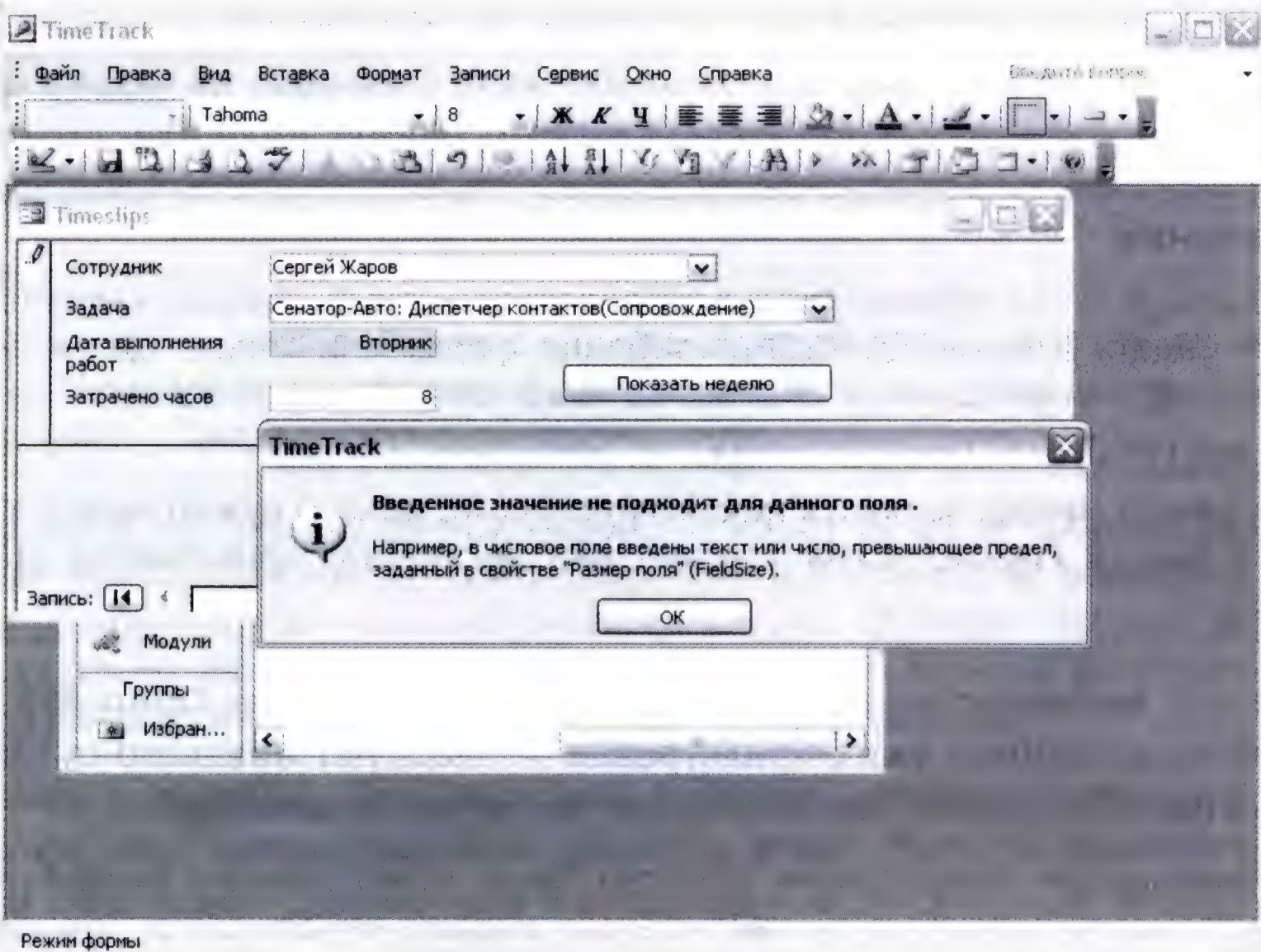


Рис. 10.5. Ошибка ввода данных, отображаемая механизмом Jet

2. Дважды нажмите клавишу <Esc>, чтобы отменить произведенные изменения и переключиться в режим конструктора.

3. В модуль формы добавьте следующую процедуру:

```
Private Sub Form_Error(DataErr As Integer, _
                        Response As Integer)
    ' Обработка ошибок механизма Jet
    Select Case DataErr
        Case 2113
            ' Ввод данных некорректного типа
            MsgBox "Введенные вами данные не подходят " & _
                "для этого поля. Попробуйте еще раз или " & _
                "для отмены нажмите клавишу <Esc>.", vbInformation
            ' Отмена стандартного сообщения об ошибке
            Response = acDataErrContinue
        Case Else
            ' Все остальные ошибки мы поручаем
            ' обрабатывать приложению Access
            Response = acDataErrDisplay
    End Select
End Sub
```

4. Сохраните форму и вызовите ее.

5. Начинайте ввод новой записи. В качестве даты введите слово "Вторник" и нажмите клавишу табуляции. На этот раз появится сообщение об ошибке, показанное на рис. 10.6. Щелкните на кнопке ОК, и вы увидите, что стандартное сообщение об ошибке, генерируемое приложением Access, не отображается.

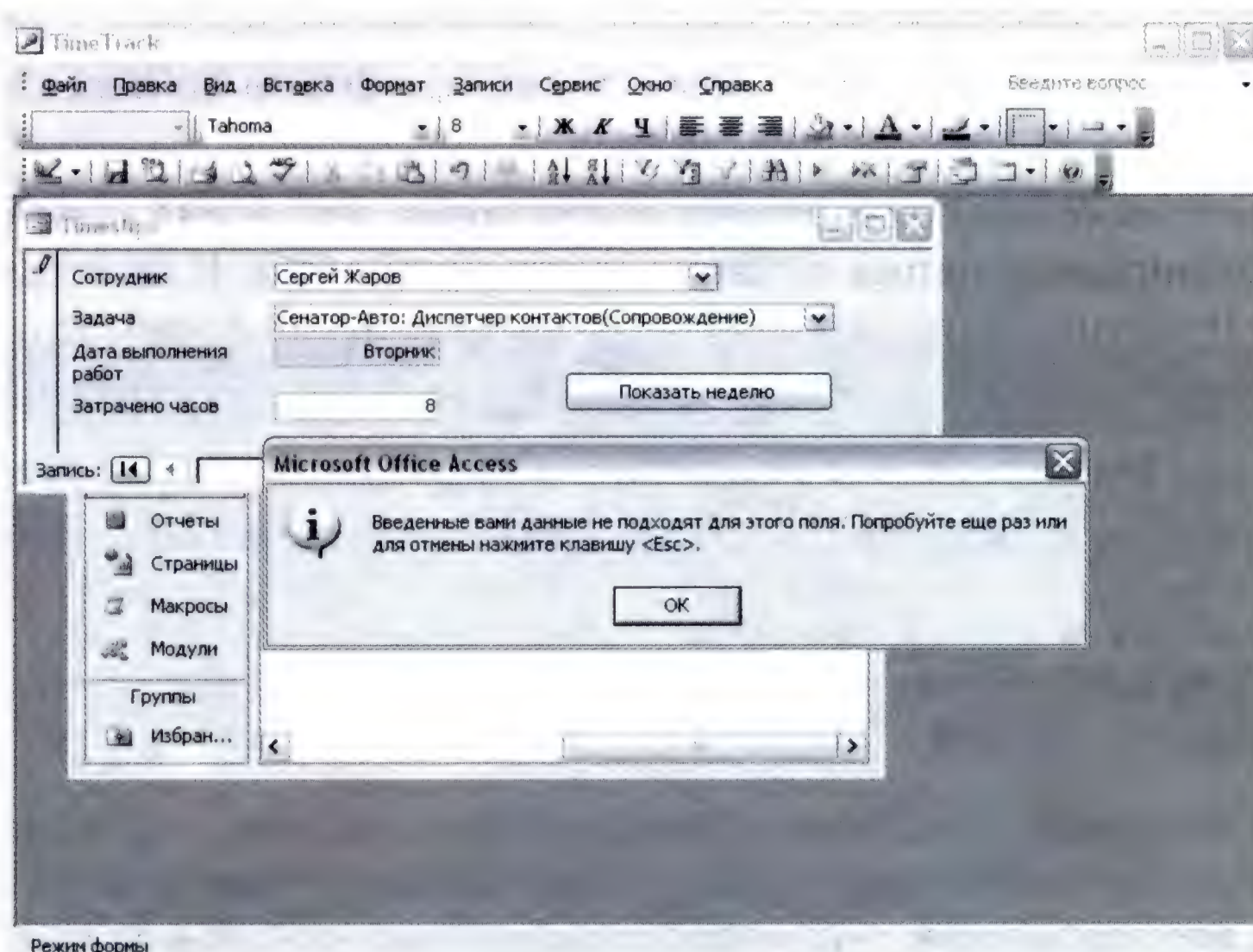


Рис. 10.6. Сообщение об ошибке типа данных, отображаемое процедурой обработки ошибок Form_Error

Если во время ввода данных механизм Jet встречает ошибку, Access вызывает обработчик события `Error` формы в коде VBA. Эта процедура имеет два аргумента. Первый из них — номер ошибки Jet, вызвавшей событие `Error`. В данном случае мы ищем ошибку ввода данных под номером 2113. Аргумент `Response` — это то, что возвращается после обработки ошибки. Если этот аргумент установить в константу `accDataErrDisplay`, будет отображено стандартное сообщение об ошибке приложения Access; а если в `accDataErrContinue` — стандартное сообщение об ошибке отображено не будет.

Совет

Не существует единого списка ошибок механизма Jet, однако вы можете составить его сами практическим путем. Для этого установите контрольную точку в первой строке процедуры `Form_Error`, и после остановки выполнения проверьте в окне `Immediate` значение аргумента `DataErr`.

Работа с несколькими экземплярами формы

Итак, что же можно сделать в программе на языке VBA, и невозможно — в интерфейсе пользователя Access? К примеру, одновременно открыть несколько экземпляров одной и той же формы. Как вы уже знаете из главы 8, объект формы является всего лишь экземпляром класса форм. Подобно любому другому классу, этот класс позволяет создавать несколько экземпляров объекта. Однако разработчики Access решили не предоставлять конечному пользователю такой возможности.

➔ Пример программы создания одного экземпляра формы приведен в разделе “Создание собственных объектов” главы 8.

Создание нескольких экземпляров одной формы в коде VBA является не более чем расширением метода создания одного экземпляра. В качестве примера можно привести процедуру, открывающую два экземпляра формы `Project`:

```
Sub CreateTwoForms()
    ' Создание и отображение двух копий
    ' формы Projects
    Dim frm1 As New Form_Projects
    frm1.Visible = True
    frm1.Move 0, 0
    Dim frm2 As New Form_Projects
    frm2.Visible = True
    MsgBox "Для продолжения щелкните OK"
End Sub
```

В этой процедуре объявляются две переменные объекта, каждая из которых ссылается на свой экземпляр формы. В этой процедуре первая форма перемещается к верхнему левому углу рабочей области Access, иначе она была бы впоследствии полностью перекрыта вторым экземпляром. На рис. 10.7 показан результат выполнения этой процедуры.

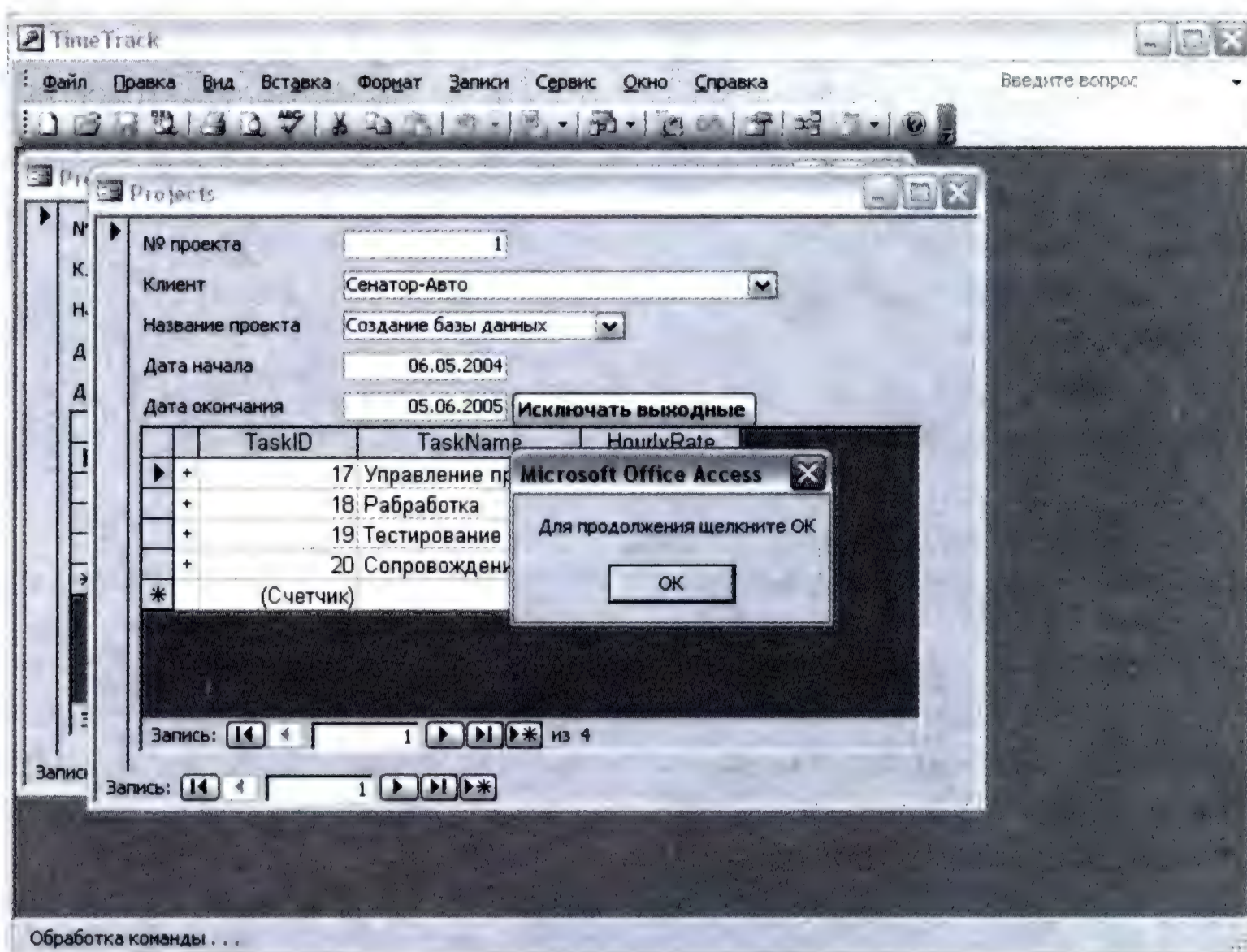


Рис. 10.7. В Access открыты два экземпляра формы Projects

Несколько экземпляров одной и той же формы ведут себя так же, как и любая отдельная форма: в них можно устанавливать свойства, перемещаться в наборе данных и т.д. Каждый экземпляр формы является отдельным членом коллекции Forms. В то же время, чтобы иметь возможность отображения формы на экране и работы с ней, каждому из экземпляров нужно выделить собственную переменную объекта.

➔ Информация о доступности переменных вне тела процедуры содержится в разделе “Время жизни переменных и констант” главы 9.

Работа с двумя экземплярами одной и той же формы

В этом практикуме мы рассмотрим практическое использование создания двух экземпляров одной формы, а также вопрос переключения между разными источниками данных. В базе данных примеров TimeTrack различные формы не очень хорошо интегрируются друг с другом. К примеру, если одновременно открыть формы Clients и Projects, каждая из них отобразит все записи соответствующих таблиц. Теперь предположим, что требуется показать только проекты определенного клиента. Для этого форму Projects нужно построить на основе запроса, который передается в качестве параметра из формы Clients. Применяя прием, показанный в практикуме, вы сможете открывать столько окон со списками проектов, сколько вам потребуется.

Для начала создадим запрос, отбирающий из таблицы проекты только конкретного клиента. Для этого откройте новый запрос в режиме конструктора, переключитесь в представление SQL и введите следующее выражение:

```
SELECT Projects.*
FROM Projects
WHERE (((Projects.ClientID)=[Forms]![Clients]![ClientID]))
ORDER BY Projects.ProjectName;
```

Сохраните этот запрос под именем `ProjectsForClient`. Теперь можно слегка модернизировать программу формы `Projects`, добавив в нее новую процедуру:

```
Public Sub LoadClient()
    ' Вызывается из клиентской формы для изменения
    ' источника записей формы
    Me.RecordSource = "ProjectsForClient"
    Me.Refresh
End Sub
```

Помните, что общая процедура класса становится методом класса — в данном случае методом формы. Это значит, что процедуру открытия формы можно вызывать и из другой формы.

Метод `Refresh` этой формы позволяет обновить записи, полученные из базы данных. Это может понадобиться в тот момент, когда изменяется источник данных и требуется отобразить уже новые данные.

Учитывая изменения, выполненные в форме `Projects`, можно изменить и форму `Clients`. Добавьте в нее новую командную кнопку с именем `cmdProjects`. Теперь введите фрагмент программы, вызываемый при щелчке на этой кнопке:

```
Dim arrProjectForms() As Form_Projects
Dim intProjectFormsCount As Integer
Private Sub cmdProject_Click()
    ' Открываем форму Projects, отображая в ней
    ' проекты текущего клиента

    ' Добавляем в массив новую переменную
    intProjectFormsCount = intProjectFormsCount + 1
    ReDim Preserve arrProjectForms( _
        1 To intProjectFormsCount)
    ' Создаем новую форму
    Set arrProjectForms(intProjectFormsCount) = _
        New Form_Projects
    ' Открываем новую форму
    arrProjectForms(intProjectFormsCount).LoadClient
    ' и отображаем еще один ее экземпляр
    arrProjectForms(intProjectFormsCount).Visible = True
End Sub
```

Первые две инструкции, которые вы видите в приведенном листинге, следует ввести в области общих объявлений модуля формы. В этой программе показано, как отслеживать произвольное количество переменных форм: для этого их нужно поместить в массив. Каждый раз при щелчке пользователя на кнопке будет создаваться новый экземпляр формы, ссылка на который будет добавляться в массив. После этого вызывается метод `LoadClient` этой формы, устанавливается ее источник данных (`RecordSource`) и форма отображается на экране.

→ Функции работы с массивами рассматривались в главе 7.

Таким образом, в форме `Clients` теперь можно выделить некоторого клиента и с помощью кнопки `cmdProjects` отобразить только его проекты. Не закрывая форму проектов, можно выделить еще одного клиента и снова отобразить его проекты. На рис. 10.8 показан результат открытия двух списков проектов, соответствующих разным клиентам.

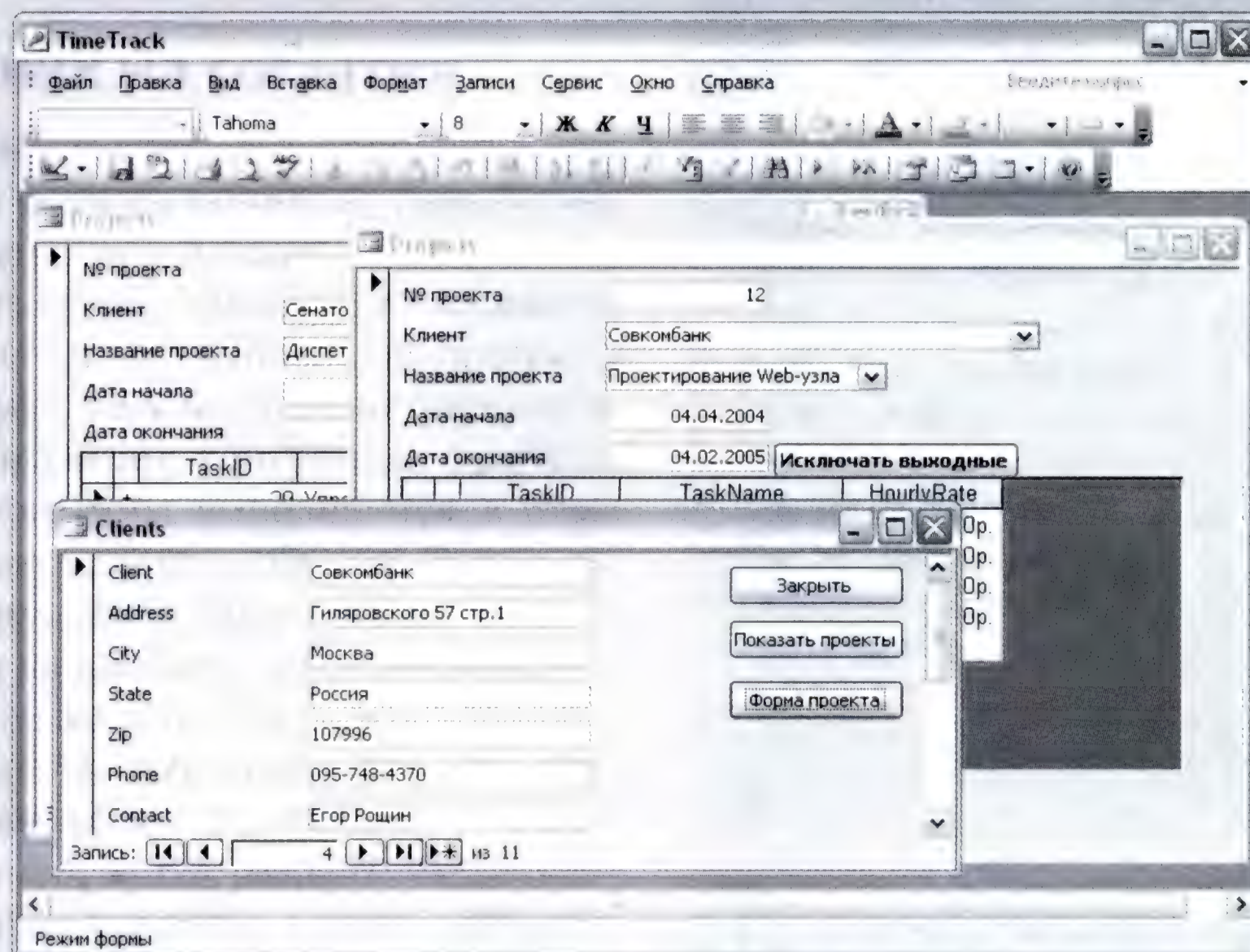


Рис. 10.8. Использование нескольких экземпляров дочерней формы

Когда форма `Clients` закрывается, массив выходит за свою область видимости. В результате все дочерние формы `Projects` закрываются также.

Анализ модели событий Access

11

Отклик на события

Одна из ключевых концепций, которую должны понимать разработчики программ для Access, состоит в том, что эта среда *управляется событиями*. Это значит, что выполнение программ на языке VBA подчинено определенным правилам: конкретные фрагменты программы выполняются в ответ на происходящие события. Этими событиями могут быть щелчки пользователя на кнопках, переходы в формах к другой записи и т.д.

Вполне вероятно, что вы уже сталкивались с событиями в Access, например, когда разрабатывали собственные макросы. Однако вопросы работы с событиями настолько глобальны, что правильным будет начать их рассмотрение с самого начала. Необходимо принять во внимание тот факт, что события происходят в определенной последовательности, и каждое из них имеет собственную цель.

Перед тем как переходить к более конкретным вопросам, нужно ясно себе представлять, как отдельные составляющие вписываются в общую концепцию. Для этого в качестве примера возьмем кнопку Открыть отчет в форме BillingReport-Setup. Если открыть эту форму в режиме конструктора, то можно увидеть, что имя этой кнопки — cmd-

В ЭТОЙ ГЛАВЕ

Отклик на события.....	195
Порядок наступления событий в элементах управления	197
События, связанные с данными	200
Последовательность событий в формах.....	202
Последовательность событий в отчетах	205
Отмена событий	206

OpenReport. Откройте окно свойств этой кнопки и вы увидите, что свойство **Нажатие кнопки** установлено в значение [Процедура обработки событий]. Это значение сообщает Access, что события щелчка на этой кнопке нужно передавать для обработки коду VBA.

Совет

Если значение свойства события отличается от [Процедура обработки событий], им может быть имя макроса, вызываемого при наступлении этого события. В данном случае при возникновении события код VBA не вызывается.

Если обработкой события занимается процедура VBA, то чтобы разобраться более детально с этим вопросом, вам потребуется загрузить редактор VBE. В нашем примере для этого достаточно перейти в строку **Нажатие кнопки** в окне свойств объекта и щелкнуть на кнопке с тремя точками рядом со значением [Процедура обработки событий]. При этом откроется редактор VBE и вы увидите в модуле Form_BillingReportSetup следующие строки:

```
Private Sub cmdOpenReport_Click()
    On Error GoTo HandleErr
    DoCmd.OpenReport "BillingReport", acViewPreview, , , _
        chkSummary
ExitHere:
    Exit Sub
HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description & " в cmdBillingReport_Click"
    Resume ExitHere
End Sub
```

Для того чтобы код VBA обрабатывал некоторое событие, необходимо выполнение следующих условий:

- В соответствующем свойстве объекта должно быть установлено значение [Процедура обработки событий].
- Текст процедуры обработки события должен находиться в корректном модуле (т.е. присоединенном к конкретной форме или отчету, в котором возникает данное событие).
- Имя процедуры должно следовать следующему шаблону:
имя_объекта.имя_события.

Совет

Если выбрать в окне свойств объекта значение [Процедура обработки событий], то при вызове редактора VBA приложение Access автоматически сформирует каркас процедуры.

Для некоторых событий, с которыми мы столкнемся далее в этой главе, существует еще одно условие: если событие имеет аргументы, их же должна в своем объявлении содержать и процедура его обработки.

Порядок наступления событий в элементах управления

Первый набор событий, который необходимо изучить, — это события, ассоциированные с элементами управления (кнопками, комбинированными списками и т.п.). Вначале рассмотрим общие вопросы переключения фокуса и изменения данных, а затем уже перейдем к событиям, специфичным для конкретных элементов управления.

Примечание

Приведенный в этой главе список событий не является исчерпывающим. Существует ряд событий, имеющих весьма ограниченное применение на практике. К примеру, несмотря на то, что при перемещении мыши в пределах текстового поля возникает соответствующее событие, большинство приложений на его никак не реагируют. Это событие в настоящей главе будет рассмотрено в качестве примера.

События переключения фокуса

По мере перемещения пользователя между элементами формы с помощью щелчков мыши или нажатий клавиши табуляции возникает масса событий. Этими событиями можно воспользоваться, чтобы, к примеру, выделить соответствующие элементы на экране, изменить некоторые внутренние значения или выполнить какие-либо другие задачи.

В этом контексте отмечается четыре ключевых события¹:

- **Вход (Enter).** Возникает сразу перед поступлением фокуса к одному элементу управления от другого в пределах одной и той же формы.
- **Выход (Exit).** Возникает сразу перед передачей фокуса от одного элемента управления к другому в пределах одной и той же формы.
- **Получение фокуса (GotFocus).** Возникает при получении элементом фокуса.
- **Потеря фокуса (LostFocus).** Возникает при потере элементом фокуса.

Совет

Понятие *фокуса* связано с активизацией некоторого элемента управления. В зависимости от типа конкретного элемента при получении фокуса в элементе отображается мигающий курсор или точечное подчеркивание.

Для того чтобы исследовать порядок возникновения событий, введем в форму `Clients` следующий текст программы. Для его ввода откройте форму

¹ Поскольку русифицированная версия приложения Access не имеет аналога в редакторе VBE, названия событий в окне свойств (русскоязычные) отличаются от тех, которые участвуют в формировании имен процедур их обработки (англоязычные). Во избежание путаницы здесь и далее будут приводиться оба варианта названия событий. — *Примеч. ред.*

Clients в режиме конструктора, после чего откройте модуль формы. (При желании вы можете воспользоваться готовым модулем формы, содержащимся в базе данных примеров TimeTrack).

```
Private Sub Client_Enter()  
    Debug.Print "Открытие поля Клиент"  
End Sub  
  
Private Sub Client_Exit(Cancel As Integer)  
    Debug.Print "Закрытие поля Клиент"  
End Sub  
  
Private Sub Client_GotFocus()  
    Debug.Print "Получение фокуса полем Клиент"  
End Sub  
  
Private Sub Client_LostFocus()  
    Debug.Print "Потеря фокуса полем Клиент"  
End Sub  
  
Private Sub Address_Enter()  
    Debug.Print "Вход в поле адреса"  
End Sub  
  
Private Sub Address_Exit(Cancel As Integer)  
    Debug.Print "Выход из поля адреса"  
End Sub  
  
Private Sub Address_GotFocus()  
    Debug.Print "Получение фокуса полем адреса"  
End Sub  
  
Private Sub Address_LostFocus()  
    Debug.Print "Потеря фокуса полем адреса"  
End Sub
```

Цель ввода этого фрагмента программы состоит в отображении в окне Immediate при возникновении событий соответствующего текста. Разместите на экране окна приложений таким образом, чтобы видеть одновременно и рабочую область Access, и редактор VBE. Теперь откройте форму Clients, и вы увидите, что два события произошли одновременно (рис. 11.1).

Как только открывается форма, фокус должен перейти к какому-либо ее элементу. Как правило, он автоматически устанавливается в объект, находящийся на первом месте в последовательности перехода. В данном случае событие входа (Enter) текстового поля Клиент (Client) наступило перед его же событием получения фокуса (GotFocus).

Нажмите клавишу табуляции для перехода в следующее по порядку поле — Адрес. В окне Immediate отобразится реакция сразу на четыре события:

```
Закрытие поля Клиент  
Потеря фокуса полем Клиент  
Вход в поле Адрес  
Получение фокуса полем Адрес
```

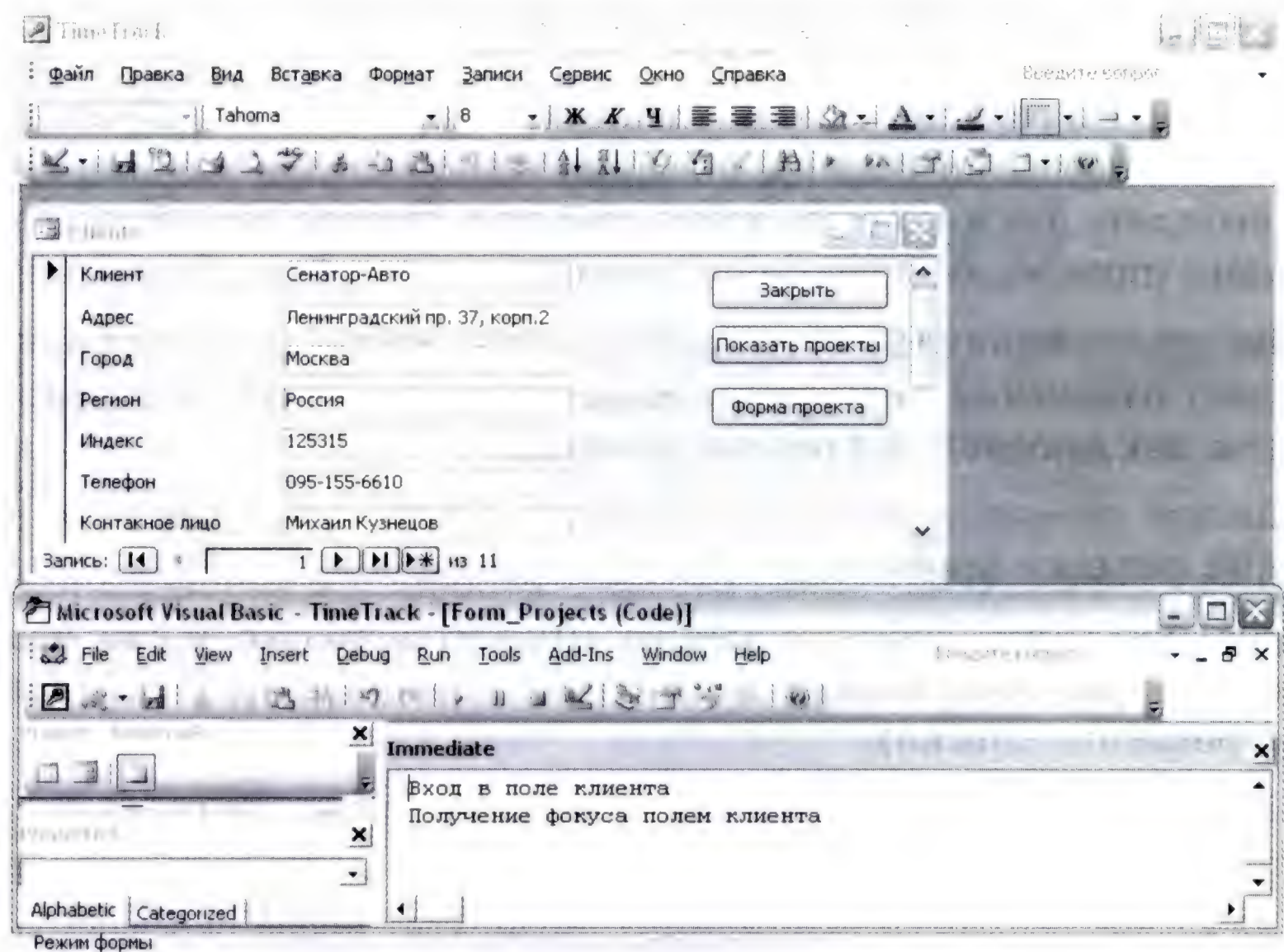



Рис. 11.1. Отслеживание порядка возникновения событий

Активизируйте теперь какую-либо другую форму в базе данных TimeTrack, к примеру, Switchboard, после чего снова вернитесь к форме Clients. В окне Immediate вы увидите реакцию на произошедшие события:

Потеря фокуса полем Адрес
Получение фокуса полем Адрес

В данном случае события выхода из поля адреса и входа в него не наступали, так как они ассоциированы с перемещением между элементами только в пределах одной формы. В то же время, в каждый конкретный момент фокус может находиться только в одной форме. Вследствие этого фокус был потерян полем адреса, а затем вернулся в него же.

И наконец, переключитесь совсем в другое приложение, например, в Excel или Блокнот, после чего снова вернитесь в Access. Вы обнаружите, что в результате этой операции события потери и получения фокуса полем адреса не наступали. Причина заключается в том, что события Access наступают *исключительно* в пределах этого приложения и никак не связаны с тем, что происходит в других компьютерных программах.

Примечание

Событие Выход (Exit) представляет собой пример события, которое может быть отменено. Этому вопросу будет посвящен отдельный раздел настоящей главы.

События, связанные с данными

Так как Access работает с базами данных, совершенно не удивительно, что множество его событий связаны с вводом пользователем данных. При этом можно выделить два множества таких событий. Первое множество связано с элементами управления, присоединенными к полям таблиц базы данных.

- До обновления (BeforeUpdate). Это событие наступает при каком-либо изменении пользователем данных в элементе управления перед тем, как данные будут переданы в форму.
- После обновления (AfterUpdate). Возникает после того, как данные попали в форму.

Второе множество событий данных относится только к тем элементам управления, в которые данные вносятся непосредственно (текстовые поля, комбинированные списки).

- Клавиша вниз (KeyDown). Возникает при нажатии пользователем на клавишу.
- Нажатие клавиши (KeyPress). Возникает после передачи значения клавиши в Access.
- Внесены изменения (Dirty). Возникает при изменении данных в форме (в полосе выбора записи отображается значок карандаша).
- Изменения (Change). Возникает при отображении символа, набранного на клавиатуре, в поле элемента управления.
- Клавиша вверх (KeyUp). Возникает, когда пользователь отпускает клавишу.

Для того чтобы увидеть эти события в действии, введите в модуль следующий фрагмент программы, посвященный текстовому полю Телефон (Phone) формы Clients:

```
Private Sub Phone_AfterUpdate()  
    Debug.Print "После обновления поля телефона"  
End Sub  
  
Private Sub Phone_BeforeUpdate(Cancel As Integer)  
    Debug.Print "До обновления поля телефона"  
End Sub  
  
Private Sub Phone_Change()  
    Debug.Print "Регистрация изменения в поле телефона"  
End Sub  
  
Private Sub Phone_Dirty(Cancel As Integer)  
    Debug.Print "Изменение пользователем номера телефона"  
End Sub  
  
Private Sub Phone_KeyDown(KeyCode As Integer, Shift As Integer)  
    Debug.Print "Клавиша вниз в поле телефона"
```



```

    Debug.Print "    Сканкод клавиши = " & CStr(KeyCode)
    Debug.Print "    Shift = " & CStr(Shift)
End Sub

Private Sub Phone_KeyPress(KeyAscii As Integer)
    Debug.Print "Нажатие клавиши в поле телефона"
    Debug.Print "    Код клавиши = " & CStr(KeyAscii)
End Sub

Private Sub Phone_KeyUp(KeyCode As Integer, Shift As Integer)
    Debug.Print "Клавиша вверх в поле телефона"
    Debug.Print "    Сканкод клавиши = " & CStr(KeyCode)
    Debug.Print "    Shift = " & CStr(Shift)
End Sub

```

Сохраните форму и попробуйте ввести какой-либо символ в ее поле телефона. В окне Immediate будет отображен следующий порядок событий (в примере была нажата клавиша а):

```

Клавиша вниз в поле телефона
    Сканкод клавиши = 70
    Shift = 0
Нажатие клавиши в поле телефона
    Код клавиши = 1072
Изменение пользователем номера телефона
Регистрация изменения в поле телефона
Клавиша вверх в поле телефона
    Сканкод клавиши = 70
    Shift = 0

```

Попробуйте нажать на другие клавиши и вы увидите аналогичную реакцию на каждое нажатие, за исключением “Изменение пользователем номера телефона”. И это вполне логично, так как сам факт изменения регистрируется только один раз. Теперь перейдите в какое-либо другое поле, и вы увидите отображение следующих событий:

```

До обновления поля телефона
После обновления поля телефона

```

Вы, наверное, заметили, что при выполнении процедур обработки событий Клавиша вниз, Нажатие клавиши и Клавиша вверх происходит некоторое дублирование данных. Вместо того чтобы программировать события для каждой из клавиш клавиатуры, разработчики Access в эти процедуры передают следующие параметры. В процедуры обработки событий Клавиша вниз и Клавиша вверх передаются сканкоды соответствующих клавиш (аргумент KeyCode) и факт нажатия клавиши <Shift>, <Alt>, <Ctrl> или какой-либо их комбинации (аргумент Shift). В процедуру обработки события Нажатие клавиши передается код ASCII, соответствующий комбинации нажатых клавиш. Более подробно о процессе декодирования значений, передаваемых событием Нажатие клавиши, вы можете узнать в справке редактора VBE.

События, специфичные для элементов управления

Некоторые элементы имеют события, которые специфичны только для них. Хорошим примером в этом направлении служит элемент командной кнопки — его событие `Нажатие кнопки (Click)` мы уже использовали в одной из предыдущих глав. Это событие возникает при щелчке мышью на командных кнопках и, как правило, программируется в формах.

Элементы выключателей, переключателей и флажков также имеют событие `нажатия кнопки`, однако здесь следует обратить внимание на следующее обстоятельство: это событие генерируется только в том случае, когда такой элемент является частью самой формы, а не группы элементов (что встречается гораздо чаще). В последнем случае событие генерируется не отдельными элементами группы, а объектом самой родительской группы.

Комбинированные списки имеют особое событие: `Отсутствие в списке (NotInList)`. Это событие возникает в том случае, когда в текстовое поле комбинированного списка пользователь вводит значение, пока отсутствующее в списковой части элемента. Это событие можно использовать для автоматизации добавления нового значения в списковую часть. Схематически это можно сделать следующим образом:

1. Создайте процедуру обработки события `Отсутствие в списке (NotInList)`.
2. Запросите пользователя, намерен ли он добавить в список новое значение.
3. Если ответ будет положительным, напишите программу добавления нового значения в таблицу, связанную с данным комбинированным списком.
4. Обновите комбинированный список.

Более детально эти операции будут рассмотрены в главе 12, посвященной особенностям работы с обычными и комбинированными списками. Событие `Отсутствие в списке` возникает после события `Клавиша вверх`.

Последовательность событий в формах

Комбинирование элементов в формы добавляет еще один уровень сложности. Формы имеют свои собственные события и они иногда пересекаются с событиями содержащихся в них элементов. Здесь очень важно знать порядок возникновения событий. Для этого можно написать процедуры обработки событий, выводящие сообщения в окно `Immediate`, и немного поработать с формой. Ввиду простоты таких процедур в книге не будет приводиться их текст, будут описаны только события. Подобные процедуры вы без особого труда сможете составить сами.

События навигации

Когда открывается форма (например, с помощью метода `DoCmd.OpenForm`), возникает последовательность из пяти разных событий:

1. Открытие (`Open`).
2. Загрузка (`Load`).
3. Изменение размера (`Resize`).
4. Включение (`Activate`).
5. Текущая запись (`Current`).

Событие Открытие — самая ранняя точка жизненного цикла формы. Событие Загрузка наступает после загрузки в форму данных. Событие Изменение размера наступает при начальной установке и каждом последующем изменении размеров формы. Событие Включение наступает, когда в форму или ее элемент переходит фокус, а событие Текущая запись — при переходе в форме между записями таблицы.

Совет

Если одну и ту же операцию требуется выполнить для всех записей, к которым будет переходить пользователь, независимо от того, что в них находится, напишите процедуру обработки события Текущая запись.

При закрытии формы происходит более простая последовательность событий:

1. Выгрузка (`Unload`).
2. Выключение (`Deactivate`).
3. Закрытие (`Close`).

Эти события происходят, соответственно, при выгрузке данных из формы, при потере формой фокуса и при закрытии формы.

Если вы перемещаетесь от одной формы к другой, событие Выключение происходит в первой форме, а событие Включение — во второй. Следует обратить внимание, что в данном случае вы не увидите других событий формы, связанных с ее открытием (Открытие, Загрузка, Текущая запись и др.).

События работы с данными в форме

Формы имеют богатый набор событий работы с данными. В конце концов, ведь именно в формах пользователи обычно вводят, редактируют и удаляют данные. Рассмотрим каждую из этих последовательностей событий отдельно. Когда пользователь в форме переходит к новой записи и начинает ввод данных, первое же нажатие на клавишу вызывает пять событий, не считая событий нажатия на клавиши, о которых мы говорили в предыдущих разделах:

1. Текущая запись (`Current`).
2. Вход в элемент управления (`Enter`).

3. Получение фокуса элементом управления (GotFocus).
4. До вставки (BeforeInsert).
5. После вставки (AfterInsert).

На два последних события следует обратить особое внимание. Событие До вставки информирует систему, что создается новая запись и в форме уже введены некоторые данные, которые должны быть сохранены. В процедуре обработки этого события данные можно изменить еще до того, как они будут сохранены в таблице. Событие После вставки является уведомлением о том, что запись была успешно создана и она уже содержит сохраненные данные.

Вы уже знаете о событиях, возникающих в элементах управления во время редактирования данных (До обновления и После обновления). По мере перемещения внутри одной записи и редактирования данных в отдельных элементах управления каждый из них инициирует эти события. После этого, когда вы пытаетесь сохранить данные в записи (выбирая в меню пункт Записи⇒Сохранить запись (Records⇒Save Record) или просто переходя к другой записи), возникают события До вставки и После вставки. Как и в других парах событий До и После, первое из них позволяет внести в данные изменения до их сохранения, а второе является индикатором успешного сохранения.

При удалении целой записи Access генерирует три события:

1. Удаление (Delete).
2. До подтверждения Del (BeforeDelConfirm).
3. После подтверждения Del (AfterDelConfirm).

Каждое из этих событий служит определенной цели. Первое из них возникает, когда пользователь подает команду на удаление записи (к примеру, щелкая на полосе выбора записи и нажимая клавишу <Del>), однако до операции реального удаления. Второе событие возникает непосредственно перед открытием приложением диалогового окна подтверждения удаления записи. В этот момент еще возможно отменить операцию удаления записи и прервать эту цепочку событий. В случае же подтверждения удаления возникает событие После подтверждения Del, знаменующее собой завершение операции.

За кулисами: буферы данных

Во время редактирования данных в форме данные хранятся в нескольких областях памяти (называемых *буферами*). С помощью диаграммы на рис. 11.2 мы попытаемся отследить жизненный цикл данных в процессе редактирования. Отображенные на ней блоки показывают буферы, в которых хранятся данные, а стрелки между блоками указывают на потоки данных.

Когда в форме вы впервые переходите к некоторой записи, ее данные извлекаются из таблицы базы данных, после чего сохраняются в буфере, содержащем только одну запись. После этого каждый из элементов управления формы отобра-

жает данные из этого буфера. Как только в некотором элементе вы начинаете ввод, данные этого поля перемещаются во второй буфер.

Если вы решили перейти к другому элементу с помощью клавиши табуляции, данные из буфера элемента перемещаются назад, в буфер формы, вызывая события До обновления и После обновления. Если же вы решили выйти из элемента с помощью клавиши <Esc>, изменения, выполненные в буфере элемента, отменяются, а сам буфер очищается. В данном случае оба события обновления не возникают.

Во время редактирования данных в элементах управления вся изменяемая запись остается в буфере формы — в этот момент ни одно из изменений не записывается в базу данных. Если дважды нажать клавишу <Esc>, все изменения в буфере формы будут игнорированы. С другой стороны, если сохранить запись, события До обновления и После обновления формы наступят, а сами измененные данные будут записаны в таблицу базы данных.



Рис. 11.2. Процесс редактирования и сохранения данных

Последовательность событий в отчетах

Отчеты также имеют свои события, хотя их состав немного беднее, чем в формах. Причина заключается в том, что пользователь не имеет возможности взаимодействовать с отображаемыми в отчете данными. При открытии отчета приложение Access генерирует два события:

- Открытие (Open);
- Включение (Activate).

Как и в случае форм, эти события представляют исходную точку, в которой отчет открыт для запуска программы, и точку, в которой устанавливается фокус в окне отчета. При закрытии формы возникает обратная последовательность событий:

- Отключение (Deactivate);
- Закрытие (Close).

Естественно, между открытием и закрытием отчета происходят и другие события, поскольку Access в это время отображает данные. Для каждого из разделов отчета последовательность событий имеет следующий вид:

- Форматирование раздела (Format);
- Возврат (Retreat);
- Отсутствуют данные (NoData);
- Печать раздела (Print).

Событие форматирования наступает в тот момент, когда Access выделяет данные, которые относятся к определенному разделу отчета, однако до реального форматирования этих данных. Событие возврата наступает, если Access принял решение “совершить откат” и переформатировать предыдущий раздел. К примеру, если вы установили свойство помещения данных на одной странице, системе может понадобиться скорректировать форматирование всех предыдущих разделов, чтобы это условие соблюсти. Событие отсутствия данных возникает в том случае, когда Access не нашел данных, которые могут быть помещены в этом разделе. Событие же печати возникает по завершению форматирования данных, но до вывода отчета на экран или принтер.

В заключение следует упомянуть еще событие Страница (Page), которое возникает сразу перед переходом программы к выводу новой страницы.

Отмена событий

В некоторых случаях может потребоваться исключить события из программы. При этом действие, вызываемое событием, также выполнено не будет. К примеру, можно из элемента управления исключить событие Выход. Какие именно события следует исключить, вы можете указать явным образом, поскольку процедуры их обработки содержат аргумент `Cancel` (Отмена). Для отмены события этот аргумент нужно установить в значение `True`, как в следующем примере:

```
Private Sub Client_Exit (Cancel As Integer)
    Cancel = True
End Sub
```

Что же случится, если в элементе отменить событие выхода? Ответ на этот вопрос зависит от действия, в результате которого возникло данное событие. Если пользователь нажал клавишу табуляции для перехода к следующему полю, курсор не сдвинется с места. Если попытаться щелкнуть мышью в каком-либо другом месте, опять же ничего не произойдет. Итак, становится очевидным, что данное событие просто так отменять нельзя. Обычно разработчики программируют какую-либо логическую проверку данных или логичности действий пользователя, и только после этого принимают решение об отмене события.

Еще одним множеством событий, которые можно отменить, являются события До (До обновления, До вставки, До подтверждения Del). Отмена любого из этих событий приостанавливает соответствующее действие. К примеру, если отменить событие До обновления, само обновление не произойдет — данные останутся в своем исходном состоянии. Это может оказаться полезным в случае, когда нужно выполнить последнюю проверку корректности и согласованности данных перед их переносом из буфера в базу данных.

ПРАКТИКУМ**Проверка данных перед их сохранением**

Итак, выполним в приложении TimeTrack дополнительную проверку данных. Для примера остановимся на форме Projects (Проекты). Двумя ключевыми моментами этой формы являются даты начала и планируемого завершения проекта. Очевидно, что дата начала должна предшествовать дате конца. В предлагаемом практикуме будет показано, как изменить логику программы для проверки совместимости значений перед тем, как их поместить в базу данных. Естественно, для этой цели можно использовать и проверку на уровне таблицы, однако в этом случае данные не будут проверяться до тех пор, пока пользователь не попытается сохранить всю запись.

Чтобы добавить описанную выше проверку корректности, откройте форму Projects в режиме конструктора и перейдите к модулю формы. Введите следующий фрагмент программы:

```
Private Sub StartDate_Exit(Cancel As Integer)
' Проверка логичности дат
If DatesAreBad Then
    MsgBox "Дата начала должна предшествовать дате завершения", _
        vbCritical
    Cancel = True
End If
End Sub

Private Sub EstimatedEndDate_Exit(Cancel As Integer)
' Проверка логичности дат
If DatesAreBad Then
    MsgBox "Дата начала должна предшествовать дате завершения", _
        vbCritical
    Cancel = True
End If
End Sub

Private Function DatesAreBad() As Boolean
' Проверка того, что дата начала
' предшествует дате завершения
On Error GoTo HandleErr

' Все в порядке
DatesAreBad = False

Dim dtSDate As Date
Dim dtEDate As Date

' Проверка на содержание в текстовых полях дат
dtSDate = CDate(StartDate.Value)
dtEDate = CDate(EstimatedEndDate.Value)

' Проверка на существование проблем
If dtEDate - dtSDate < 0 Then
```



```
        DatesAreBad = True
    End If

ExitHere:
    Exit Function

HandleErr:
    ' В случае любой ошибки считаем, что даты некорректны
    DatesAreBad = True
    Resume ExitHere
End Function
```

В этом фрагменте программы введена обработка событий Выход (Exit) двух полей — даты начала проекта и даты его завершения. Следует отметить, что изменение значения любого из них может вызвать проблему. Так как проверка проводится в двух полях, то имеет смысл вынести саму логику проверки в отдельную функцию и вызывать ее из процедур обработки событий. В самой этой функции вычисляется разность между датами завершения и начала. Если результатом является положительное число, функция возвращает False; в противном случае — True.

В каждой из процедур обработки событий, если функция DateAreBad возвращает True, то выполняются следующие действия. Вначале выводится сообщение о характере ошибки, а затем значение аргумента Cancel устанавливается в True, что обязует пользователя ввести в поле допустимое значение.

Работа с простыми и комбинированными списками

12

Заполнение элементов списка

Элементы управления являются интерфейсными объектами, реализующими взаимодействие пользователя и приложения за счет отображения данных и ввода информации. Простые и комбинированные списки позволяют отображать список позиций, из которых пользователь может выбрать одну или несколько.

Вы, по всей вероятности, уже пользовались специальным мастером для наполнения списковых элементов, однако такое решение задачи ограничивает содержимое списков. Иногда требуется большая гибкость — позволяющая при необходимости изменять содержимое списков “на лету”.

Списковые элементы имеют два общих свойства, которыми можно манипулировать программным образом и управлять содержимым списков.

- **Тип источника строк** (Row Source Type). В этом свойстве определяется, что будет служить источником данных списка: таблица, запрос, фиксированный список значений или список полей таблицы или запроса.

В ЭТОЙ ГЛАВЕ

Заполнение элементов списка.....	209
Добавление и не добавление в список.....	215
Множественный выбор	222
Функции обратного вызова	225

- **Источник строк (Row Source).** В зависимости от типа источника в этом свойстве определяется тип данных, отображаемых в данном элементе. Существует три типа источников данных.
 - Если тип установлен в значение *Таблица* или *запрос*, в свойстве *Источник строк* следует задать имя таблицы или запроса либо выражение SQL.
 - Если тип установлен в значение *Список значений*, в параметре *Источник строк* в явном виде этот список должен быть указан. В качестве разделителя отдельных значений можно использовать символ точки с запятой.
 - Если тип установлен в значение *Список полей*, источник строк опять-таки должен быть таблицей, запросом или выражением SQL. Вряд ли вам придется использовать такую установку, однако в списке типов она все равно присутствует, так как некоторые из мастеров, которые сами по себе являются формами Access (использующими код VBA), основаны именно на этом типе.

Примечание

В настоящей главе описана работа с двумя списковыми элементами в коде VBA. При этом предполагается, что вы уже знакомы с общим назначением и характеристиками обоих элементов.

По всей вероятности, вы уже знакомы с этими тремя свойствами, так как устанавливали их в окне свойств элементов. В то же время установить их можно и в коде VBA, что позволит динамически изменять состав списков.

Примечание

В настоящий момент нам понадобится не только свойство *Данные* (Control-Source), так как для программного взаимодействия с элементом совсем не обязательно, чтобы он был подчиненным или свободным. Вряд ли вам придется использовать код VBA для управления списком таких элементов.

Несмотря на то, что, теоретически, списковые элементы можно наполнить данными в произвольный момент, самым подходящим для этого местом является момент загрузки формы или установки фокуса на данный элемент. В любом случае вы должны будете установить свойство типа источника строк (RowSourceType) в некоторое значение, используя следующий синтаксис:

```
элемент.RowSourceType = значение
```

где *элемент* является именем элемента управления, а *значение* — одним из строковых значений, перечисленных в табл. 12.1. Для установки свойства источника строк (RowSource) используется следующий синтаксис:

```
элемент.RowSource = источник_данных
```

где *источником_данных* может служить таблица, запрос, выражение SQL или список значений, в зависимости от типа источника данных.

Таблица 12.1. Значения типов источников строк в VBA

Установка	Строка VBA
Таблица/запрос	"Table/Query"
Список значений	"Value List"
Список полей	"Field List"

Простой элемент с фильтрованным списком

Рассмотрим простой пример создания фильтрованного комбинированного списка в форме Employees. В частности, мы добавим комбинированный список в заголовок формы, после чего с помощью кода VBA наполним его списком сотрудников. (В данном примере использован комбинированный список, но вы можете проделать то же самое и с элементом простого списка — синтаксис и состав свойств в них практически идентичен, однако комбинированный список занимает на форме меньше места.)

Для начала откроем форму Employees в режиме конструктора. Перейдем к заголовку формы и вставим в него элемент комбинированного списка. Присвойте ему имя cboFilter и задайте отображаемое имя — Искать. После этого выполните следующие действия:

- 1. Откройте модуль формы. Введите следующий фрагмент, в котором присваиваются начальные значения двум свойствам формы:

```
Private Sub Form_Open(Cancel As Integer)
    cboFilter.RowSourceType = "Table/Query"
    cboFilter.RowSource = "Employees"
End Sub
```

- 2. Переключитесь в представление формы и откройте список нового элемента (рис. 12.1).

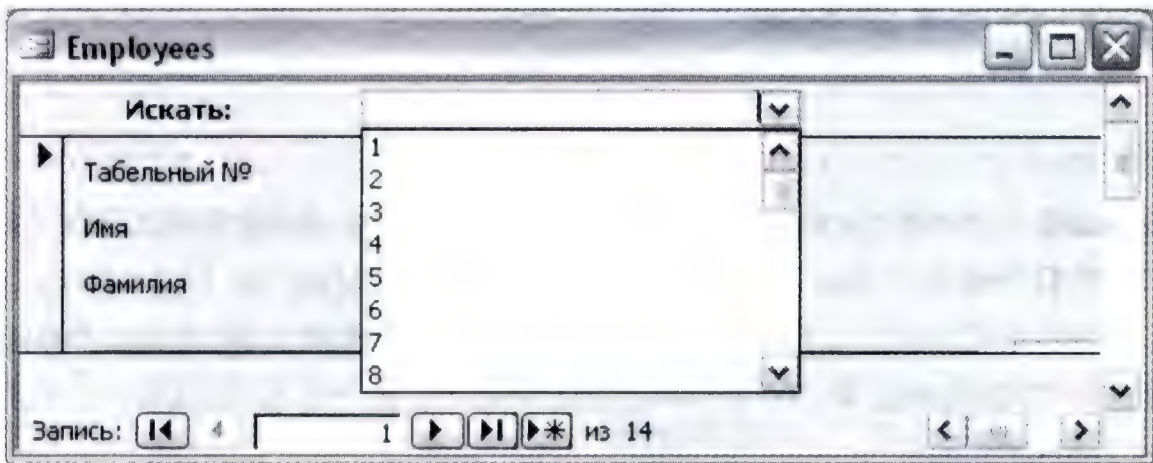


Рис. 12.1. Комбинированный список отображает содержимое первого столбца таблицы Employees

Результат наверняка обманет ваши ожидания. По умолчанию в комбинированном списке отображается только один столбец из источника данных — первый.

При использовании кода VBA свойство ColumnCount следует установить в тот номер столбца, данные которого должны отображаться в списке; при этом свойству ColumnWidth следует присвоить значение ширины столбца (рис. 12.2):

```
cboFilter.ColumnCount = 3
cboFilter.ColumnWidths = "0"
```

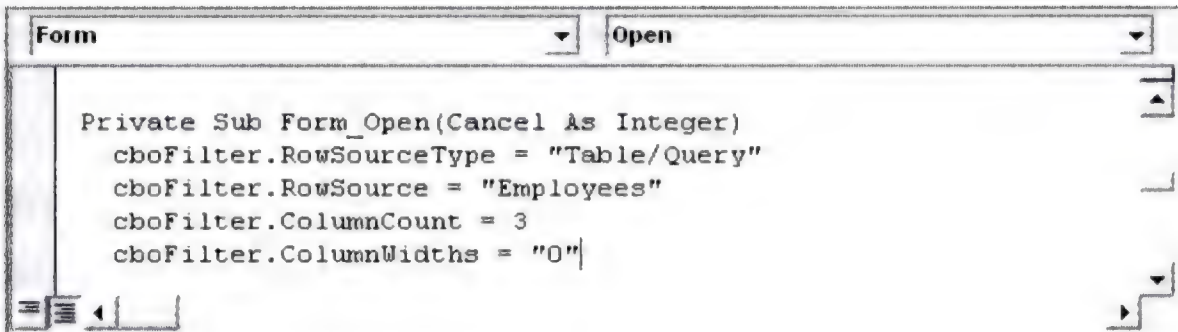


Рис. 12.2. Для манипуляции свойствами элементов можно использовать код VBA

Сохраните введенный фрагмент программы. После этого снова откройте форму Employees, и на этот раз при обращении к новому элементу откроется список фамилий сотрудников (рис. 12.3). Обратите внимание на то, что табельные номера сотрудников, содержащиеся в первом столбце таблицы, также отображаются, просто малая ширина элемента не позволяет их увидеть.

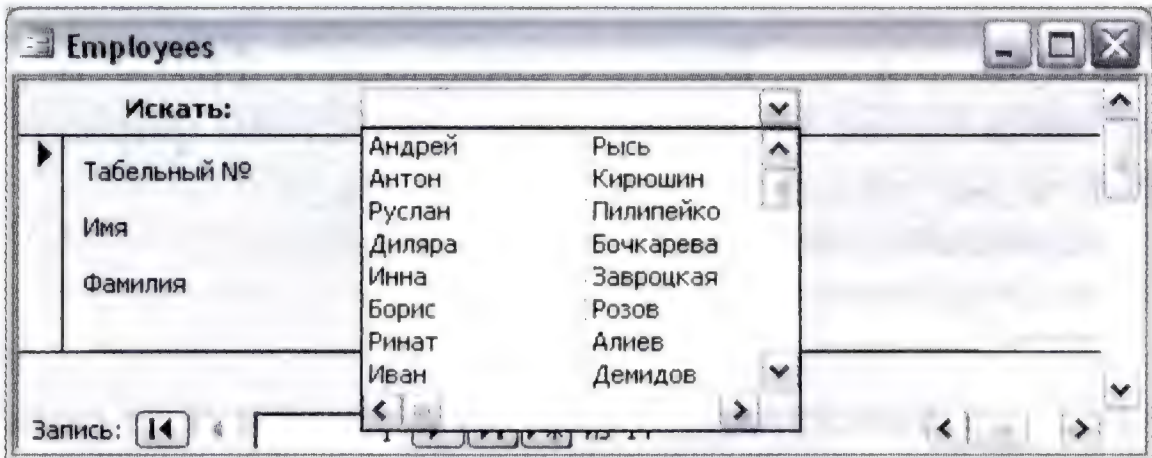


Рис. 12.3. Теперь в комбинированном списке отображаются нужные данные

Совет

Приведенный выше пример позволяет программным способом установить те свойства, которые раньше вы устанавливали вручную. Однако при этом не забывайте, что по умолчанию свойство **Присоединенный столбец** (Bound) установлено в первый столбец источника данных, а не в первый столбец отображаемого списка данных.

Только что созданный список еще не выполняет что-то отличное от простого отображения данных — например, выводит информацию о выбранном сотруднике. Для этого нужно добавить процедуру обработки события выбора элемента в списке — именно поэтому мы обращаемся к событию **Нажатие кнопки** (Click). Итак, ваши следующие действия:

1. Вернитесь в редактор VBE с помощью клавиш <Alt+Tab> или выбора соответствующего значка на панели задач Windows.
2. В модуле формы в списке объектов выберите `cboFilter`, как показано на рис. 12.4.

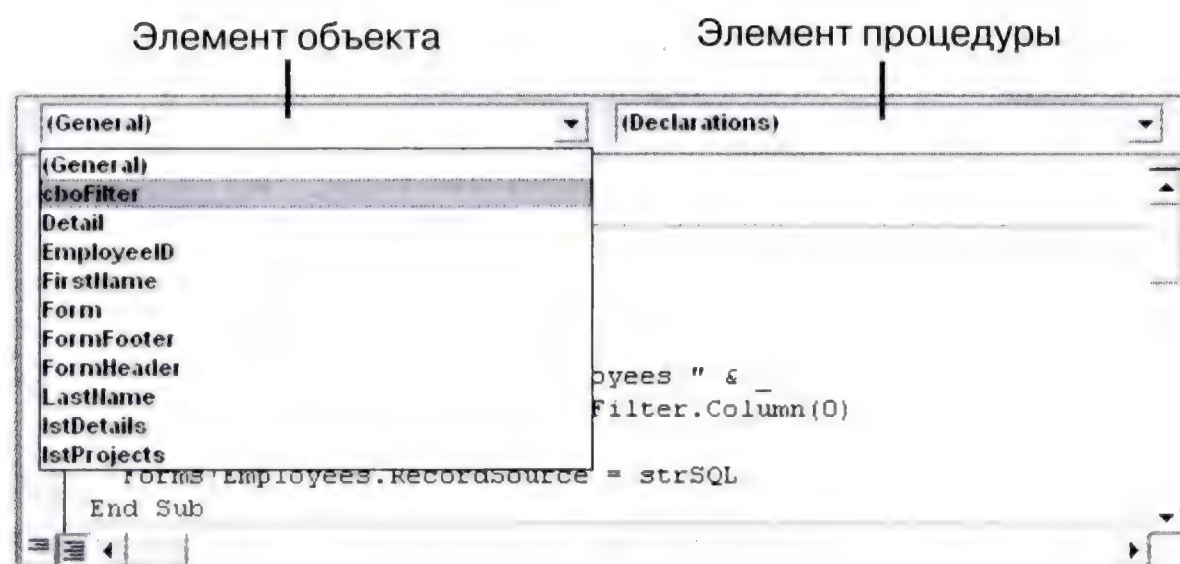


Рис. 12.4. Выберите объект в элементе списка объектов

3. Теперь в списке процедур выберите пункт `Click` (этот комбинированный список находится справа). После этого в рабочей области будет отображен скелет процедуры обработки события Нажатие кнопки (рис. 12.5).

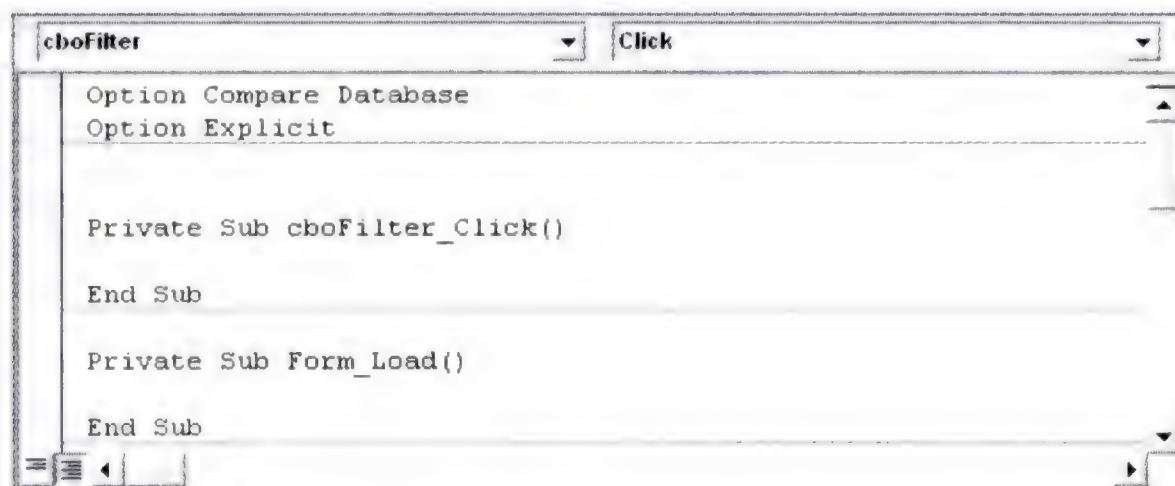


Рис. 12.5. Редактор автоматически сформировал скелет процедуры обработки события

4. Внутри скелета новой процедуры введите следующие строки (строки заголовка и окончания процедуры дублировать не нужно):

```
Private Sub cboFilter_Click()
    Dim strSQL As String
    strSQL = "SELECT * FROM Employees " & _
        "WHERE EmployeeID = " & cboFilter.Column(0)
    Debug.Print strSQL
    Forms!Employees.RecordSource = strSQL
End Sub
```

5. Откройте форму и выберите какую-либо фамилию в комбинированном списке. При этом в форме автоматически отобразится информация о данном сотруднике (рис. 12.6). Если вы переключитесь в окно редактора VBE, то в окне Immediate увидите модифицированное выражение SQL.

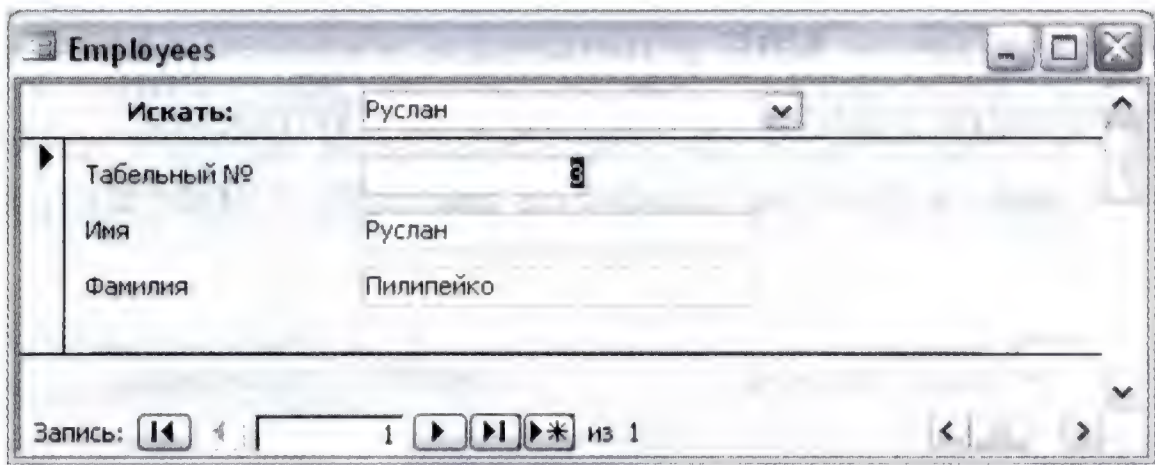


Рис. 12.6. При выборе в комбинированном списке имени сотрудника отобразилась информация из его записи в таблице Employees

Примечание

Существует масса способов ввода программ в модуль. В одном примере было выбрано свойство Вход, чтобы сформировать костяк процедуры, во втором примере для этой же цели мы использовали комбинированные списки окна модуля. Изучив основные способы ввода процедур, вы сможете решить, с какими из них будет удобнее работать. В следующих примерах будут представлены все возможные способы ввода процедур.

В предыдущем примере мы рассмотрели всего один из вариантов типов списка — Таблица или запрос. Пожалуй, он наиболее распространен на практике. Создание списка значений или полей ничем не сложнее. Ключевой особенностью здесь является задание свойства типа источника данных — нужно дважды все проверить, иначе тип источника не будет соответствовать данным, и в список попадут некорректные значения. Для примера рассмотрим следующий фрагмент программы и результат, к которому его выполнение приводит (рис. 12.7).

```
cboFilter.RowSourceType = "Field List"
cboFilter.RowSource = "Employees"
```

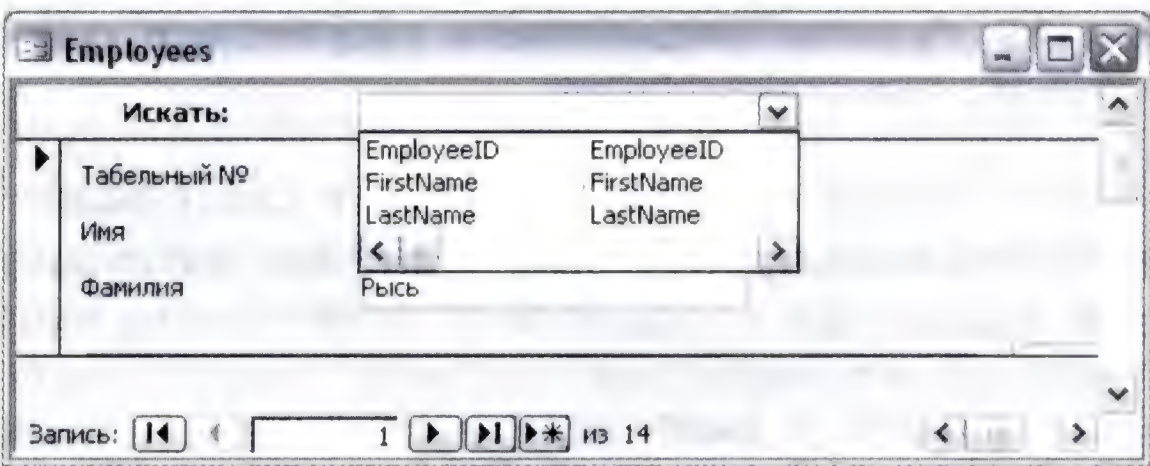


Рис. 12.7. Список полей отображает имена полей таблицы Employees

На рис. 12.8. отображен результат выполнения следующего фрагмента программы:

```
cboFilter.RowSourceType = "Value List"
cboFilter.RowSource = "Сергей;Тимур;Андрей"
cboFilter.ColumnCount = 1
```


Закомментируйте или удалите выражения присвоения значения свойству `ColumnWidths` в окне свойств элемента. Последний приведенный пример непрактичен, так как он не динамичен. Это значит, что вы не сможете в любой момент обновить свойство источника строк (`RowSource`) и таким образом изменить список элемента.

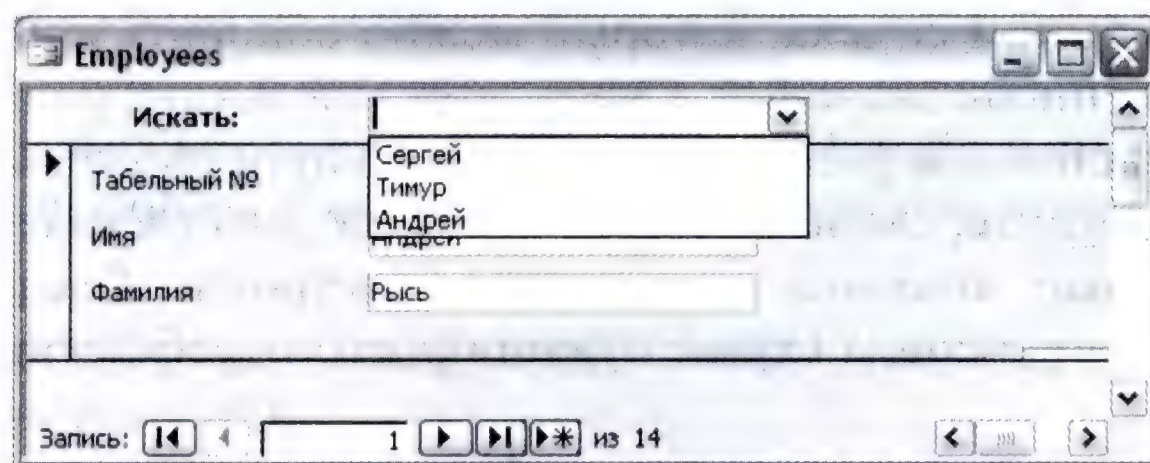


Рис. 12.8. Отображение списка predetermined значений в элементе

Добавление и не добавление в список

Элементы комбинированных списков обладают одним свойством, которого лишены обычные списки. В прикрепленном к ним текстовом поле можно ввести некоторое значение, отсутствующее в списке. По умолчанию это значение можно использовать в форме, однако в список оно не добавляется — для этого нужно выполнить некоторую дополнительную работу.

Ввод отсутствующего в списке значения инициирует событие `Отсутствие в списке` (`NotInList`). По умолчанию в ответ на это событие ничего не происходит; данное событие имеет два аргумента — `NewData` (Новые данные) и `Response` (Отклик), которые можно использовать для добавления значения в список программным способом.

Аргумент `NewData` равен введенному в текстовое поле комбинированного списка значению. Аргумент `Response` задает способ обработки события и может быть равен следующим встроенным константам:

- `acDataErrorDisplay`. Это значение принято по умолчанию и отображает системное сообщение об ошибке данных. Используйте это значение, когда не хотите предоставлять пользователю право ввода дополнительных значений.
- `acDataErrorContinue`. Пользователю отображается заданное вами сообщение. Как правило, пользователя спрашивают, не намерен ли он добавить в список новое значение. Если ответ будет положительным, позиция добавляется в список, а в тексте программы следует установить аргумент `Response` в значение `acDataErrAdded`. Если ответ отрицательный, значение аргумента `Response` устанавливается в значение `acDataErrContinue`.

- `acDataErrAdded`. Это значение аргумента означает, что введенное значение, содержащееся в аргументе `NewData`, добавлено в список элемента комбинированного списка.

Обновление списка значений

Простейшим типом списка, который можно обновить программным способом, является список значений. Существует множество подходов к этой задаче, однако в настоящем разделе мы пойдем по пути наименьшего сопротивления. Как вы помните, свойство `Источник строк` имеет в качестве своего значения разделенные точками с запятой подстроки. Следовательно, для изменения списка в данном случае нужно просто переформировать значение этого свойства.

В качестве примера мы создадим свободный элемент простого списка, с которым будет несложно работать. (Присоединенные элементы списков значений непрактичны, и поэтому используются крайне редко.) Для создания элемента, показанного на рис. 12.9, выполните следующие действия:

1. Откройте новую пустую форму в режиме конструктора и добавьте в нее элемент комбинированного списка.
2. Присвойте элементу имя `cboColors` и измените подпись на **Цвета :**.
3. Установите свойство `Тип источника строк` в значение `Список значений`.
4. Введите в свойстве `Источник строк` следующую строку:
Красный ; Белый ; Синий.
5. Сохраните форму и откройте ее в режиме формы (рис. 12.9).

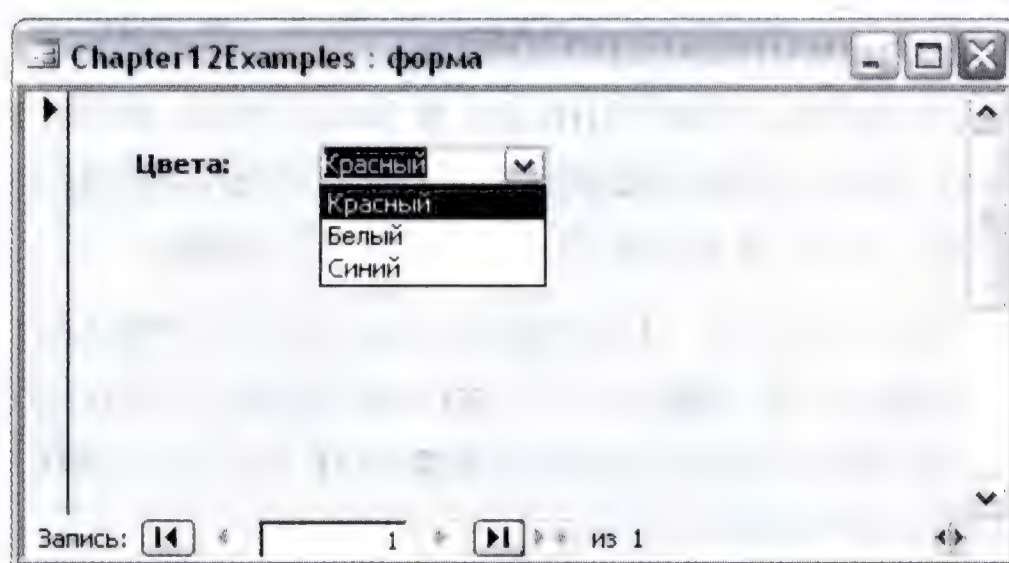


Рис. 12.9. В этом простом комбинированном списке отображается фиксированный список значений

Теперь введите цвет **Желтый** в текстовой части комбинированного списка. Программа Access позволит вам ввести это значение, однако ничего в результате не произойдет. Это значит, что в следующий раз, когда вы захотите ввести желтый цвет, это слово вам не удастся выбрать из списка — набирать его придется заново.

Добавление в список нового пункта требует написания небольшой процедуры VBA. Для этого выполните такие действия:

1. Переведите форму снова в режим конструктора и измените значение свойства Ограничиться списком (Limit to List) в Да (Yes).
2. Перейдите к событию Отсутствие в списке (On Not in List) и установите в нем значение [Процедура обработки событий] ([Event Procedure]). Щелкните на кнопке с тремя точками в этой строке. Откроется модуль формы с введенным в нем скелетом процедуры обработки события.
3. Дополните процедуру следующим текстом (разумеется, без строк заголовка и завершения):

```
Private Sub cboColors_NotInList(NewData As String, _
                                Response As Integer)
    Dim bytResponse As Byte
    bytResponse = MsgBox("Вы хотите добавить " & _
        NewData & "цвет в список?", vbYesNo)
    If bytResponse = vbYes Then
        Response = acDataErrAdded
        cboColors.RowSource = Me!cboColors.RowSource & _
            & ";" & NewData
    Else
        Response = acDataErrContinue
        Me!cboColors.Undo
    End If
End Sub
```

4. Теперь вернитесь в форму и переключитесь в режим формы.
5. Снова попробуйте ввести слово **Желтый** в текстовой части комбинированного списка.
6. После того как Access откроет окно сообщения, показанное на рис. 12.10, щелкните на кнопке Да. На рис. 12.11 изображен уже модифицированный список элемента, теперь уже желтого цвета. Открыв окно свойств этого элемента, вы увидите, что и значение свойства Источник строк изменилось и также содержит желтый цвет.

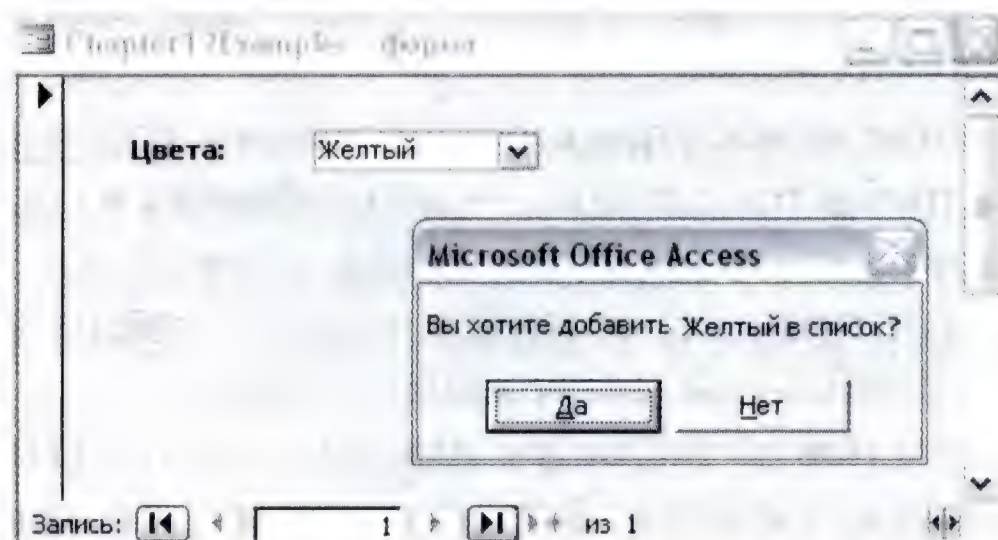


Рис. 12.10. При вводе значения, отсутствующего в списке, отображается это окно сообщения

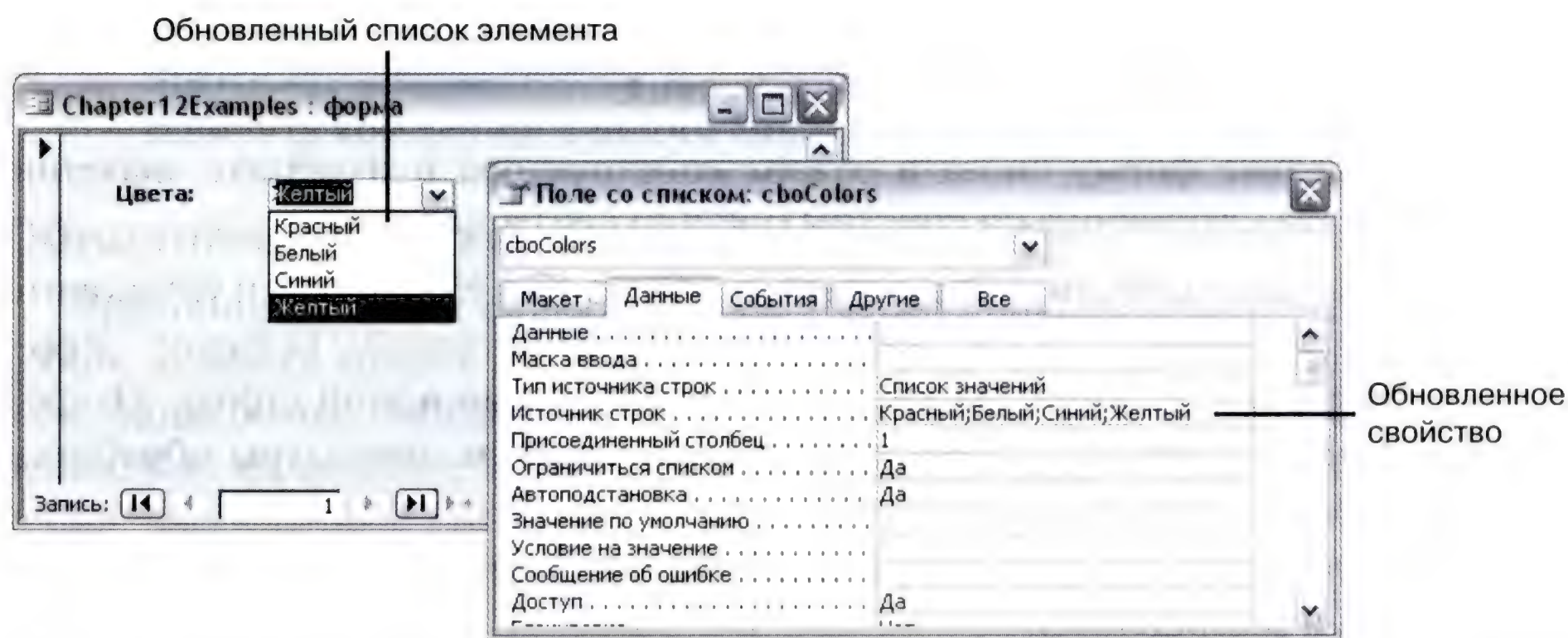


Рис. 12.11. При положительном ответе на показанное выше сообщение в список добавляется новое значение

При вводе значения, отсутствующего в списке, возникает событие **Отсутствие в списке**, процедура обработки которого открывает окно сообщения (см. рис. 12.10) и сохраняет ваш ответ в переменной `bytResponse` — в зависимости от нее в последующей конструкции `If` в список добавляется или не добавляется новое значение.

При добавлении в список новой позиции процедура изменяет значение аргумента `Response` на константу `acDataErrAdded`, что подтверждает факт реальной вставки нового значения в свойство. Обратите внимание, что в случае отказа пользователя от добавления (часть `Else` конструкции `If`) аргумент `Response` устанавливается в константу `acDataErrContinue`, после чего с помощью метода `Undo` очищается текстовая часть комбинированного списка.

Обновление списка при связи с таблицей или запросом

Списки, подчиненные запросам или таблицам, самые распространенные в списковых элементах; к тому же они являются самыми гибкими. Для того чтобы отобразить некоторую позицию в списке, ее нужно добавить в связанный источник данных (таблицу или запрос). Списки эти могут быть подчиненными и свободными; соответственно и решения задачи добавления элемента кодом VBA будут разными.

Простейшее решение относится к подчиненным спискам, потому что все, что в данном случае требуется сделать, — это добавить в соответствующий источник данных новую запись, а затем снова обратиться к этому элементу. Чтобы понять весь этот процесс, создайте новый элемент комбинированного списка и свяжите его с таблицей, содержащей данные.

Для начала создадим новую таблицу с именем `Colors` (Цвета) и всего одним текстовым полем — также `Colors`. Теперь создадим три записи и введем в них значения `Красный`, `Белый` и `Синий`. Теперь создадим на основе этой таблицы новую форму. Для этого в списке таблиц окна баз данных выделим таблицу

Colors, после чего выберем пункт меню Вставка⇒Форма (Insert⇒Form). В открывшемся окне дважды щелкнем на строке Конструктор (Design View).

В открывшуюся новую форму вставим элемент комбинированного списка, присвоим ему имя cboBound и установим его источник строк в таблицу Colors. Установим метку данного элемента в Переплет. Теперь установим свойство Источник строк в следующую строку, являющуюся запросом на языке SQL:

```
SELECT DISTINCT Colors FROM Colors ORDER BY Colors
```

Это выражение извлекает из таблицы Colors уникальные значения поля Colors и присваивает их списковой составляющей комбинированного списка.

➔ Краткий обзор выражений SQL содержится в Приложении.

В этом виде элемент комбинированного списка будет отображать все различные значения поля Colors таблицы Colors. В нем также будет выделяться текущая выбранная запись (рис. 12.12). Теперь вы можете ввести в выделенную позицию новое значение, после чего соответствующая запись может быть добавлена в таблицу Colors. Однако новая запись не сразу будет отображена в комбинированном списке — для этого потребуются снова открыть форму. После ее открытия в этом элементе будут отображены все новые уникальные значения.

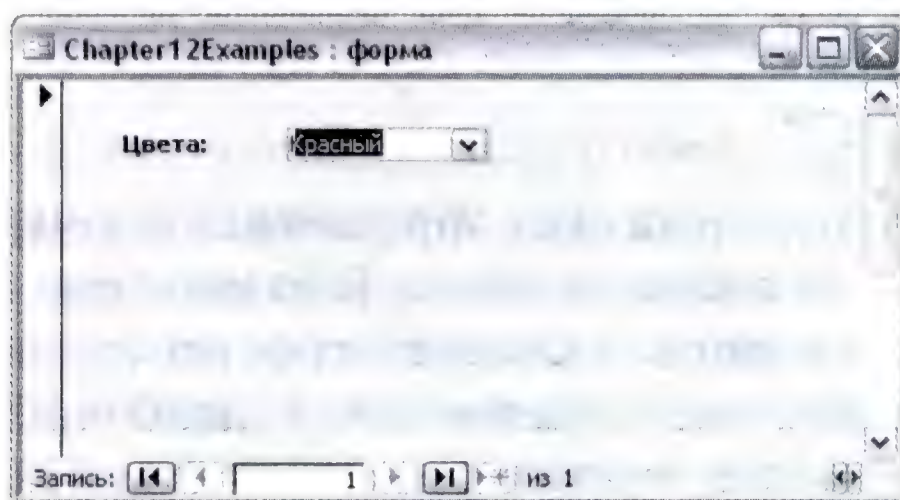


Рис. 12.12. Подчиненные элементы управления получают свои списки из присоединенных источников данных

Простейшее решение возникшей проблемы заключается в отслеживании момента пополнения таблицы данными и выполнении повторного запроса; для этого нам потребуется процедура на языке VBA. Откройте форму в режиме конструктора, дважды щелкните на элементе cboBound, после чего событие После обновления (AfterUpdate) установите в значение [Процедура обработки событий]. Щелкните на кнопке с тремя точками в строке этого события. Откроется окно модуля формы, в котором нужно ввести следующий фрагмент программы:

```
Private Sub cboBound_AfterUpdate()  
    DoCmd.RunCommand acCmdSaveRecord  
    cboBound.Requery  
End Sub
```


Совет

Метод `AddItem` используется для добавления текущего значения в фиксированный список значений элемента комбинированного списка. Если потребуется совместимость с более старыми версиями Access, имейте в виду, что этот метод несовместим с версиями, более ранними, чем Access 2002.

Теперь вернемся к форме и отобразим ее в представлении формы. Щелкните на кнопке **Новая запись** панели навигации, введите **Желтый** цвет и нажмите клавишу <Enter>. На рис. 12.13 показан новый список; в котором уже содержится название желтого цвета.

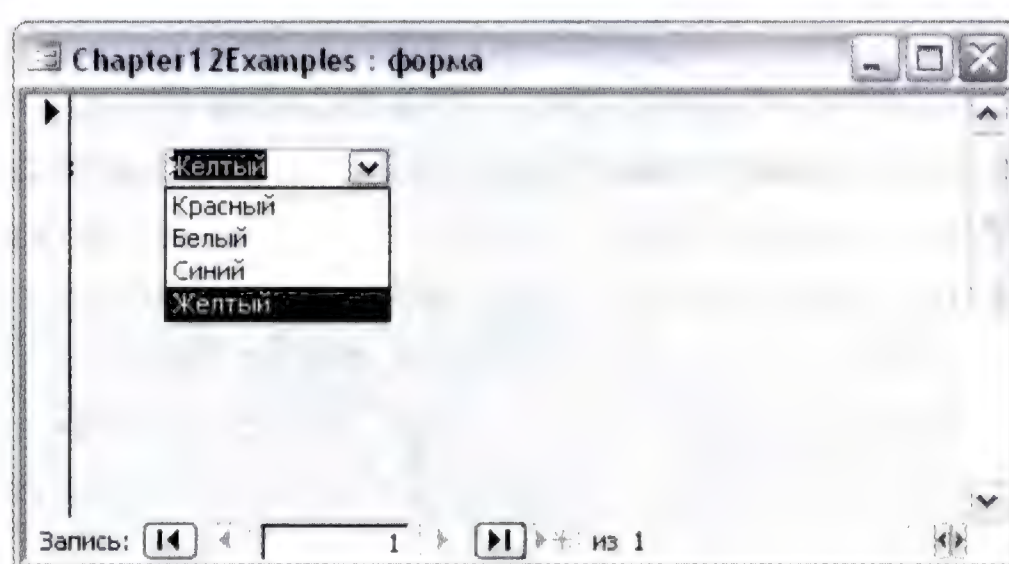


Рис. 12.13. Установите для элемента режим немедленного обновления списка при вводе нового значения

В приведенном примере решалась упрощенная задача, в которой все новые значения добавлялись в список элемента. В то же время следует помнить, что не всегда списковые элементы являются лучшим вариантом ввода данных, поскольку в них слишком высока вероятность ошибочного добавления в список нового элемента. Более мудрым решением будет использование подчиненных списковых элементов.

Список также можно подгрузить из несвязанного источника данных. Это будет гарантией того, что исходные данные не будут изменены случайно; в то же время у вас остается возможность обновления исходного источника данных (а следовательно, и списка) новыми значениями. Этот вариант будет наилучшим для случаев, когда элемент основывается на классификаторах, а не на фактических данных.

Вставьте новый комбинированный список в форму первого примера. Присвойте этому элементу имя `cboUnbound`, подпись **Свободное поле** и введите в его свойстве **Источник строк** следующий запрос:

```
SELECT DISTINCT Colors FROM Colors ORDER BY Colors
```

После этого установите значение свойства **Ограничиться списком** в Да.

Данное выражение SQL идентично подчиненному элементу комбинированного списка. В то же время поскольку данный элемент является свободным, для обновления его источника данных требуется более сложное реше-

ние. В настоящее время элемент отображает список уникальных цветов, содержащихся в таблице Colors. Если ввести значение, не содержащееся в списке, Access отвергнет его и отобразит сообщение об ошибке.

Откройте форму в режиме конструктора, дважды щелкните на элементе cboUnbound, после чего событие **Отсутствие в списке** установите в значение [Процедура обработки событий]. Вызовите окно модуля формы, щелкнув на кнопке с троеточием в этой строке, и введите следующую процедуру:

```
Private Sub cboUnbound_NotInList(NewData As String, _
                                Response As Integer)
    Dim cnn As New ADODB.Connection
    Dim strSQL As String
    Dim bytResponse As Byte
    Set cnn = CurrentProject.Connection
    bytResponse = MsgBox("Вы хотите добавить новый элемент " & _
        "в список?", vbYesNo, "Обнаружен новый элемент")
    If bytResponse = vbYes Then
        strSQL = "INSERT INTO Colors(Colors) VALUES('" & _
            NewData & "')"
        Debug.Print strSQL
        cnn.Execute strSQL
        Response = acDataErrAdded
    ElseIf bytResponse = vbNo Then
        Response = acDataErrContinue
        Me!cboUnbound.Undo
    End If
End Sub
```

Теперь вернитесь в форму и отобразите ее в представлении формы. Введите **черный** цвет в прикрепленное текстовое поле свободного элемента. Это действие вызовет возникновение события отсутствия в списке (NotInList) и выполнение соответствующей процедуры. Откроется окно сообщения, показанное на рис. 12.14. Щелкните на кнопке **Да**, и это приведет к выполнению выражения SQL INSERT INTO. Это выражение позволяет вставить текущее значение (аргумент NewData) в источник данных (таблицу Colors). На рис. 12.15 показан обновленный список. Если вы откроете теперь таблицу Colors, то там также найдете новую запись о черном цвете.

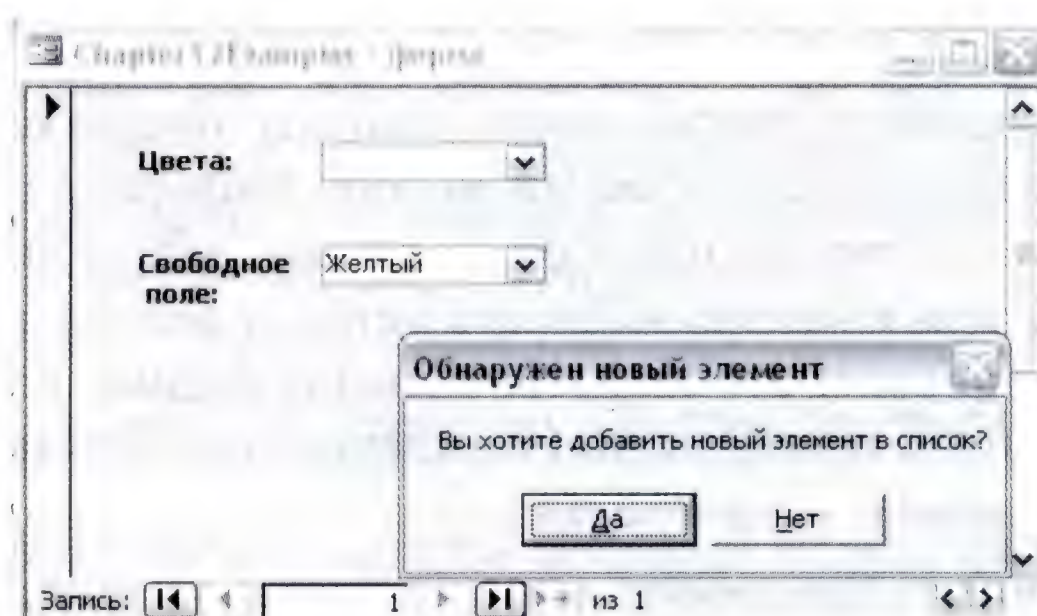


Рис. 12.14. Событие **Отсутствие в списке** привело к отображению этого сообщения

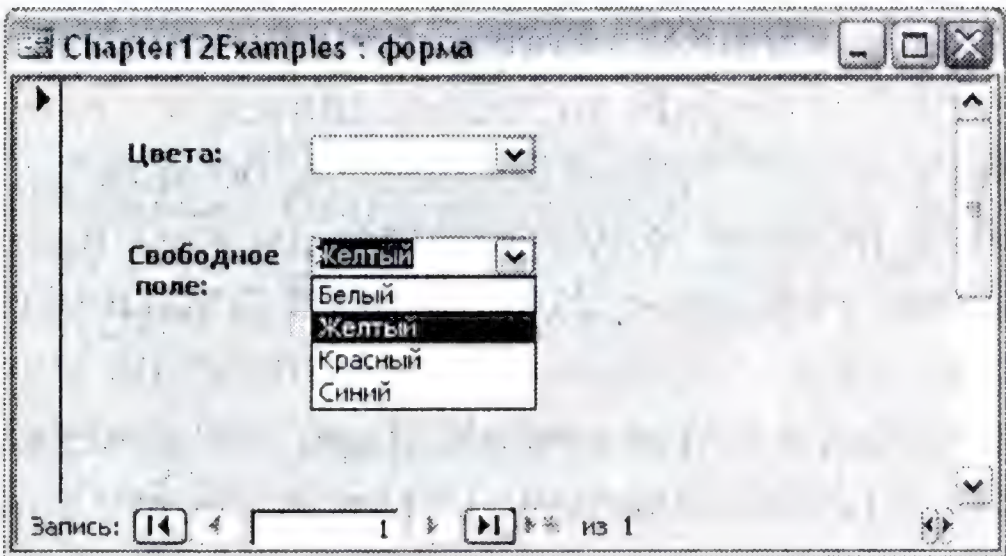


Рис. 12.15. Текущее значение было добавлено в список элемента

Помните, что вы совершенно не обязаны добавлять в список новые элементы — при отображении приложением Access показанного выше сообщения вы вправе щелкнуть на кнопке Нет. При этом значение аргумента `Response` будет установлено в константу `acDataErrContinue`, что позволит продолжить работу без внесения каких-либо изменений в источник данных. В производственных ситуациях иногда требуется вводить значения, немного отличающиеся от стандартных, но их не обязательно вносить в список для дальнейшего постоянного использования. Для упрощения данного примера в этом случае мы просто очищаем поле с помощью метода `Undo`.

Примечание

Существует еще одно решение рассматриваемой задачи — на этот раз с использованием объекта `Recordset` (Набор данных). Проще говоря, это — образ в оперативной памяти связанной таблицы. Нет ничего предосудительного в использовании этого объекта, который подробно рассматривается в главе 17. Однако следует заметить, что выражения SQL требуют написания меньшего объема программного кода и при этом работают быстрее объектов `Recordset`.

Множественный выбор

Простой список позволяет выполнять те операции, которые недоступны в комбинированном — в нем можно производить множественный выбор. По умолчанию в элементе списка можно выбрать только одну позицию. Однако изменяя значение его свойства Несвязное выделение (`MultiSelect`) в значения Простой (`Simple`) или Со связным выбором (`Extended`), вы можете разрешить пользователю выбирать в списке несколько значений.

Проще всего установить это свойство в макете формы, однако то же можно сделать и программным путем, используя следующий синтаксис:

```
имя_списка.MultiSelect = настройка
```

где *настройка* может принимать одно из трех значений, перечисленных в табл. 12.2.

Таблица 12.2. Настройка свойства множественного выбора

Установка	Описание	Числовое значение
Отсутствует (None)	Запрет множественного выделения (используется по умолчанию)	0
Простой (Simple)	Выделение и снятие выделения путем щелчка мыши или нажатия клавиши пробела	1
Со связным выбором (Extended)	Множество последовательных позиций выбирается путем удержания клавиши <Shift> и щелчков на первом и последнем элементе группы. Множество непоследовательных позиций выбирается путем удержания клавиши <Ctrl> и последовательных щелчков на каждом из элементов	2

Как определить, что выбрано, а что нет

Получить единственное выбранное значение из простого или комбинированного списка предельно легко — оно содержится в свойстве Value:

```
элемент.Value
```

На самом деле свойство Value принято в объектах элементов управления по умолчанию, поэтому его явного упоминания при обращении не требуется; однако все же рекомендуется для повышения читабельности текста программы указывать это свойство явно.

Получение значения из списков с разрешенным множественным выбором требует большей работы. Как правило, для извлечения такой информации используют конструкцию цикла For...Each — она позволяет пройтись по всем выбранным позициям в списке независимо от их количества.

➔ Более подробно конструкция For...Each описана в главе 8.

Рассмотрим простой пример списка с множественным выбором, выводящего в окно Immediate все выбранные значения с помощью цикла For...Each. Откройте форму примера со свободным элементом и вставьте в нее простой список. Присвойте ему имя lstCustomers, введите в свойстве Источник строк следующее выражение SQL:

```
SELECT Client FROM Clients
```

После этого установите свойство Несвязное выделение в значение Простой.

Любым из известных вам методов откройте модуль формы и введите следующий программный фрагмент:

```
Private Sub lstCustomers_LostFocus()  
    Dim varItem As Variant
```



```

Dim lst As Access.ListBox
Set lst = lstCustomers
' Проверка, выделен ли хоть один элемент
If lst.ItemsSelected.Count = 0 Then
    MsgBox "Пожалуйста, выберите заказчика", _
        vbOKOnly, "Ошибка"
    Exit Sub
End If
' Цикл по всем выделенным позициям,
' при этом выделение последовательно снимается
For Each varItem In lst.ItemsSelected
    Debug.Print lst.ItemData(varItem)
    lst.Selected(varItem) = 0
Next
End Sub

```

Вернитесь в форму и переключите ее в режим формы. В описанной процедуре использовано событие потери фокуса, которое возникает при выходе из списка. Что же произойдет, если не выделить ни один элемент списка? Нажмите клавишу табуляции три раза, что позволит установить фокус в интересующем нас элементе, и сразу же переместить фокус на другой элемент, ничего при этом не выделив в первом. Приложение Access откроет окно сообщения, показанное на рис. 12.16.

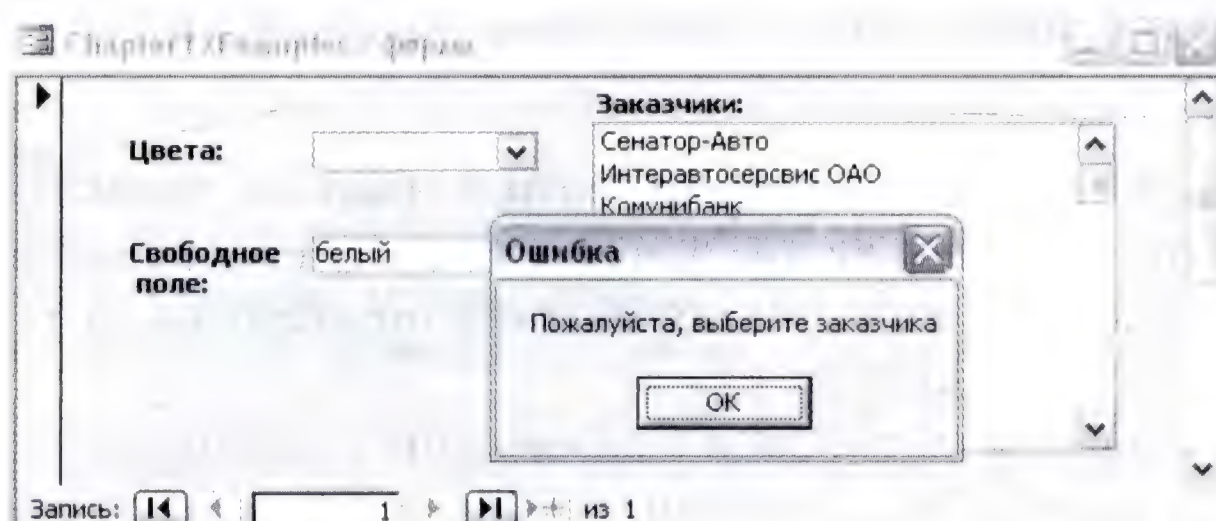


Рис. 12.16. Процедура предупреждает, что ни одна позиция списка не была выделена

Перед попыткой извлечения выбранных записей нужно убедиться, что в списке выделена хотя бы одна позиция. Оператор `If` проверяет количество элементов в коллекции `ItemsSelected`, и если оно равно нулю, это значит, что ни одна позиция списка не выделена.

Теперь вернитесь к списку и выделите первую и третью позиции (рис. 12.17). Перейдите к любому из присутствующих на форме комбинированных списков, созданных в предыдущих примерах, чтобы получить событие потери фокуса списка клиентов.

После определения того, что коллекция `ItemsSelected` не пуста, цикл `For...Each` проходит по всем ее элементам. Свойство `ItemData` содержит текст элемента, который выводится в окно `Immediate` инструкцией `Debug.Print` (рис. 12.18). После этого соответствующее свойство `Selected` устанавливается в нуль, что снимает выделение с элемента.

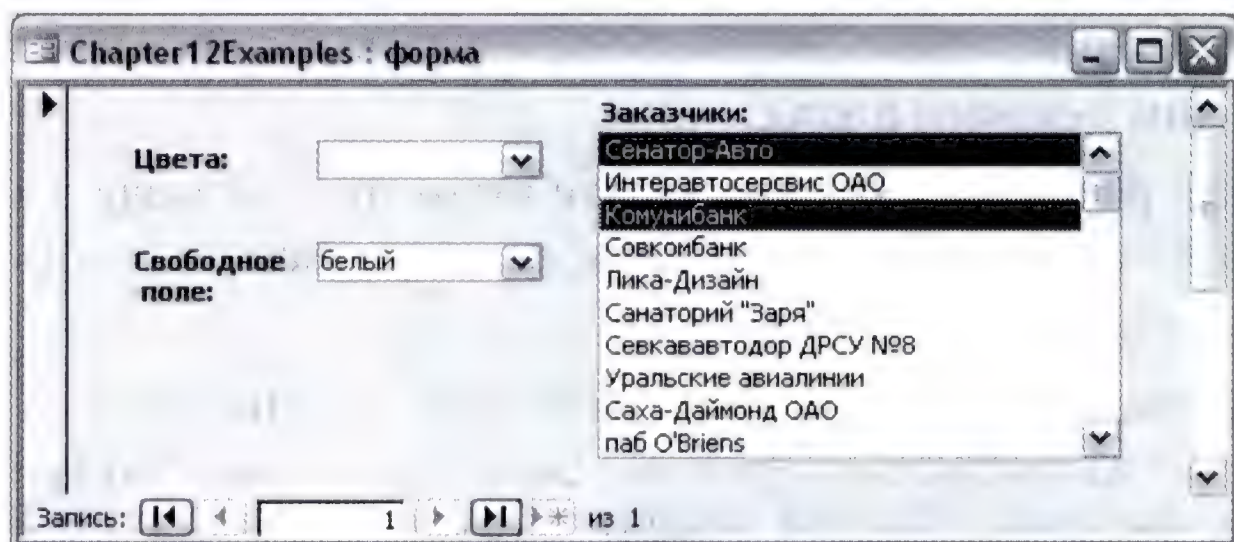


Рис. 12.17. В списке множественного выбора выделите несколько позиций



Рис. 12.18. Выделенные элементы списка выводятся в окно Immediate

Функции обратного вызова

Элемент со списком значений создать несложно, однако он имеет одно ограничение, о котором следует знать. Когда свойство Тип источника строк установлено в значение Список значений, свойство Источник строк (а это и есть реальное хранилище списка) может содержать не более 2 045 символов. В большинстве случаев этого количества вполне достаточно, однако если его не хватает, следует подумать о другом решении, например, об использовании функций обратного вызова для пополнения списка.

Для того чтобы в Access отображался список, нужно задать ряд параметров, которые устанавливаются в свойствах объекта. В частности, приложению нужно знать, сколько строк и столбцов содержится в списковом элементе. Эти значения приложению Access предоставляет *функция обратного вызова*. Эти функции аналогичны всем остальным; отличие заключается в том, что вы ссылаетесь на них в свойстве Тип источника строк объекта. К тому же функция обратного вызова использует *библиотеку DAO*, поэтому на нее должна быть установлена ссылка; в противном случае этот метод пополнения списка не будет работать. (В базе данных примеров TimeTrack.mdb ссылка на библиотеку DAO уже установлена.)

В следующем примере для пополнения элемента списка используется функция обратного вызова. Выполните такие действия:

1. Вставьте элемент простого списка в форму примера и назовите его `lstCallback`.
2. Введите в качестве типа источника строк название функции `CallbackList` (это слово вы можете набрать поверх отображаемого по

умолчанию типа Таблица/запрос). Функция еще не существует; мы ее создадим немного позже.

3. Запустите редактор VBE и создайте стандартный модуль (можете также воспользоваться модулем Chapter12 базы данных примеров TimeTrack).
4. Введите следующую функцию, после чего сохраните ее:

```
Function CallbackList(ctrl As Control, id As Variant, _
    row As Variant, col As Variant, code As Variant) As Variant
    Dim ctr As DAO.Container
    Dim dbs As DAO.Database
    Set dbs = CurrentDb
    Set ctr = dbs.Containers!Forms
    Select Case code
        Case acLBInitialize
            CallbackList = 1
        Case acLBOpen
            CallbackList = 1
        Case acLBGetRowCount
            CallbackList = ctr.Documents.Count
        Case acLBGetColumnCount
            CallbackList = 1
        Case acLBGetColumnWidth
            CallbackList = -1
        Case acLBGetValue
            CallbackList = ctr.Documents(row).Name
        Case acLBGetFormat
            CallbackList = -1
    End Select
    Set dbs = Nothing
    Set ctr = Nothing
End Function
```

Откройте форму в представлении формы. На рис. 12.19 показаны эта форма и элемент простого списка (в правом нижнем углу), отображающий список всех форм.

При открытии формы в представлении формы элемент списка обращается к функции `CallbackList`. В этот момент за кулисами происходит масса интересных событий, и вам потребуется использовать ту же структуру: передаваемые аргументы и внутренние константы, используемые в конструкции `Select Case`.

Инструкция `CurrentProject.AllForms.Count` определяет количество строк в элементе, подсчитывая число документов в коллекции `Forms`. После этого инструкция `CurrentProject.AllForms(row).Name` извлекает имя каждого элемента из коллекции `Forms`.

Функции обратного вызова являются продвинутыми и в то же время достаточно мощными инструментами. Они позволяют получить полное управление над данными, отображаемыми списковыми элементами. Приложение Access вызывает эту процедуру каждый раз, когда элементу требуется получить строку для списка, и вы можете динамически решать, что в этой строке отображать.

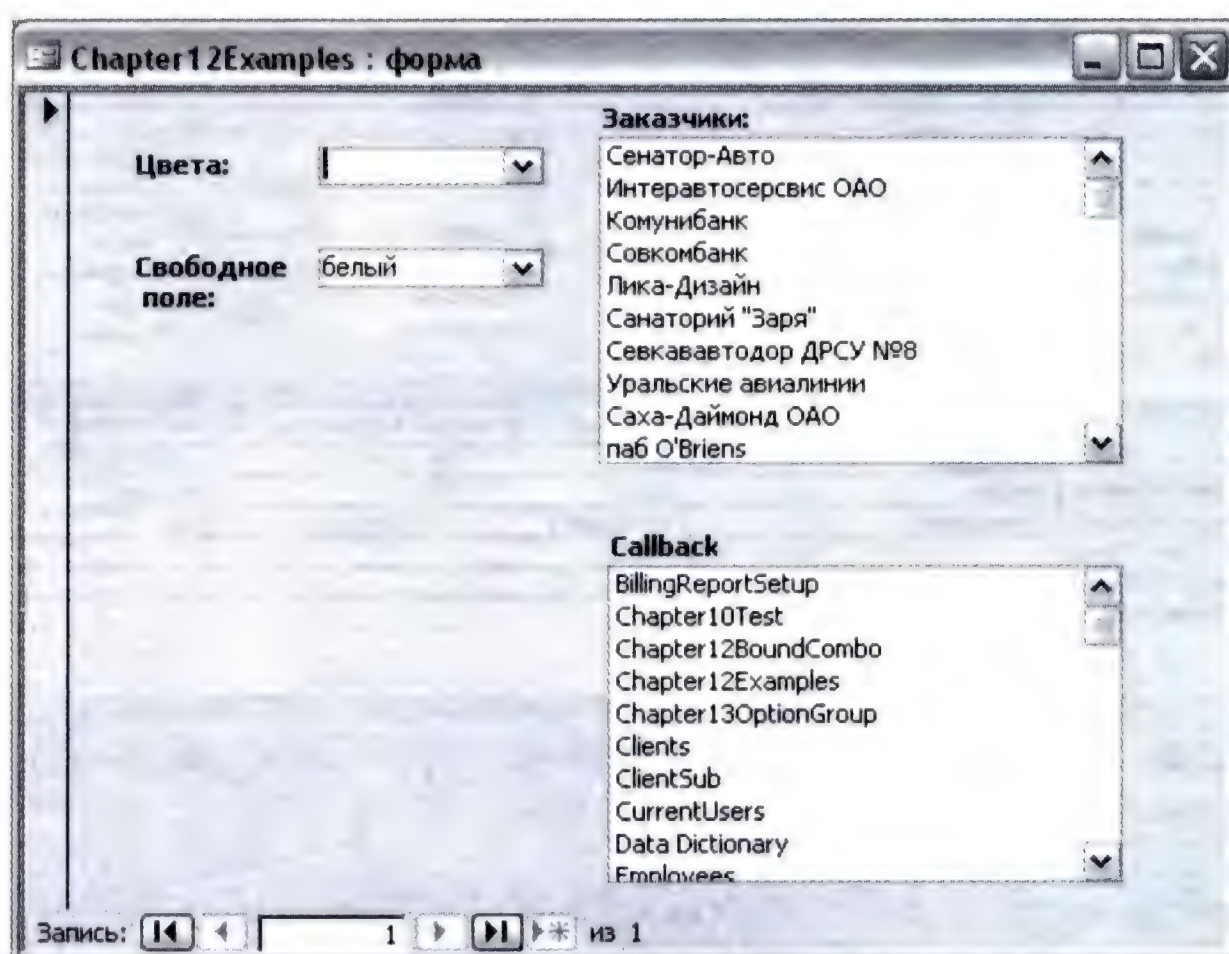


Рис. 12.19. В элементе списка для наполнения его списком форм использована функция обратного вызова

ПРАКТИКУМ

Использование простых списков в качестве списков с детализацией

В этой главе был представлен ряд примеров типичного использования списковых элементов. Комбинированные списки обладают хорошими свойствами фильтрации; в то же время простые списки отлично подходят для перечисления объектов базы данных, например, в отчетах и формах — в результате пользователь может выбрать то, что ему больше подходит в конкретной ситуации. Теперь рассмотрим нетрадиционное применение списковых элементов.

Предположим, что нам требуется использовать форму `Employees` для просмотра информации о проектах, выполняемых отдельными служащими. В данном случае можно воспользоваться табличными формами, разбитыми на подформы или открываемыми щелчком на командной кнопке. Списковые элементы в этой задаче — довольно интересная альтернатива. В данном контексте уместным будет применить то, что называют *эффектом детализации*, к списковым элементам (это не так просто сделать для подчиненных форм и табличных форм). Имеется в виду возможность получения дополнительной информации о выбранном элементе списка при щелчке или двойном щелчке на нем.

В первую очередь потребуется создать запрос, показанный на рис. 12.20. Присвоим ему имя `HoursWorkedByProject`. Обратите внимание, что в форме открыта строка группировки, так что постарайтесь заполнить ее ячейки корректными значениями. Теперь в качестве критерия столбца `EmployeeID` (Табельный номер сотрудника) введите следующую конструкцию:

```
[Forms]![Employees]![EmployeeID]
```

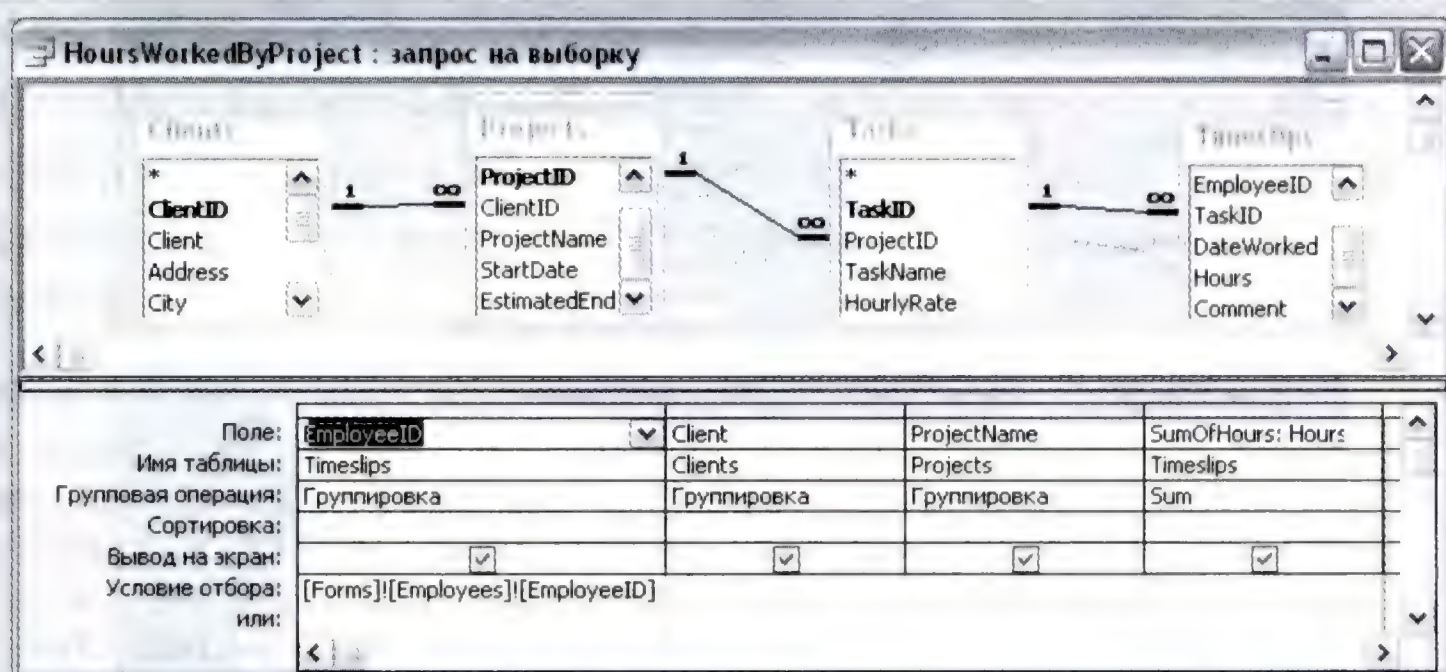



Рис. 12.20. В этом запросе суммируются часы, затраченные сотрудником на каждую из задач

Списковое поле на форме, показанной на рис. 12.21, отображает результат запроса. Это — записи, специфичные для конкретного сотрудника. В табл. 12.3 перечислены дополнительные свойства списковых полей, отображенных на данной форме. Теперь откройте модуль формы, введите приведенные здесь процедуры обработки событий и сохраните форму:

```
Private Sub Form_Current()
    lstProjects.Requery
    lstDetails.RowSource = ""
End Sub

Private Sub lstProjects_DblClick(Cancel As Integer)
    Dim strSQL As String
    strSQL = "SELECT Projects.ProjectID, Tasks.TaskName, " _
        & "Tasks.HourlyRate, Timeslips.DateWorked " _
        & "FROM (Projects INNER JOIN Tasks ON " _
        & "Projects.ProjectID=Tasks.ProjectID) " _
        & "INNER JOIN Timeslips ON Tasks.TaskID=Timeslips.TaskID " _
        & "WHERE EmployeeID = " & Forms!Employees!EmployeeID _
        & " AND ProjectName = '" & lstProjects.Column(2) & "'" _
        & "ORDER BY TaskName, ProjectName, DateWorked ASC"
    Debug.Print strSQL
    lstDetails.RowSource = strSQL
End Sub
```

Для прохождения по записям можно использовать кнопки навигации формы или элемент комбинированного списка, позволяющий фильтровать записи. При наступлении события Текущая запись формы, процедура `Form_Current` обновляет элемент списка проектов проектами указанного сотрудника за счет выполнения запроса. В этой же процедуре устанавливается источник строк по умолчанию для списка задач проекта в значение "".

Для отображения более детальной информации о проекте в левом списке нужно дважды щелкнуть на нем. Процедура обработки двойного щелчка в этом списковом элементе изменяет свойство источника строк списка задач в выражение запроса SQL, который отбирает задачи заданного сотрудника в проекте, выбранном в первом списке.

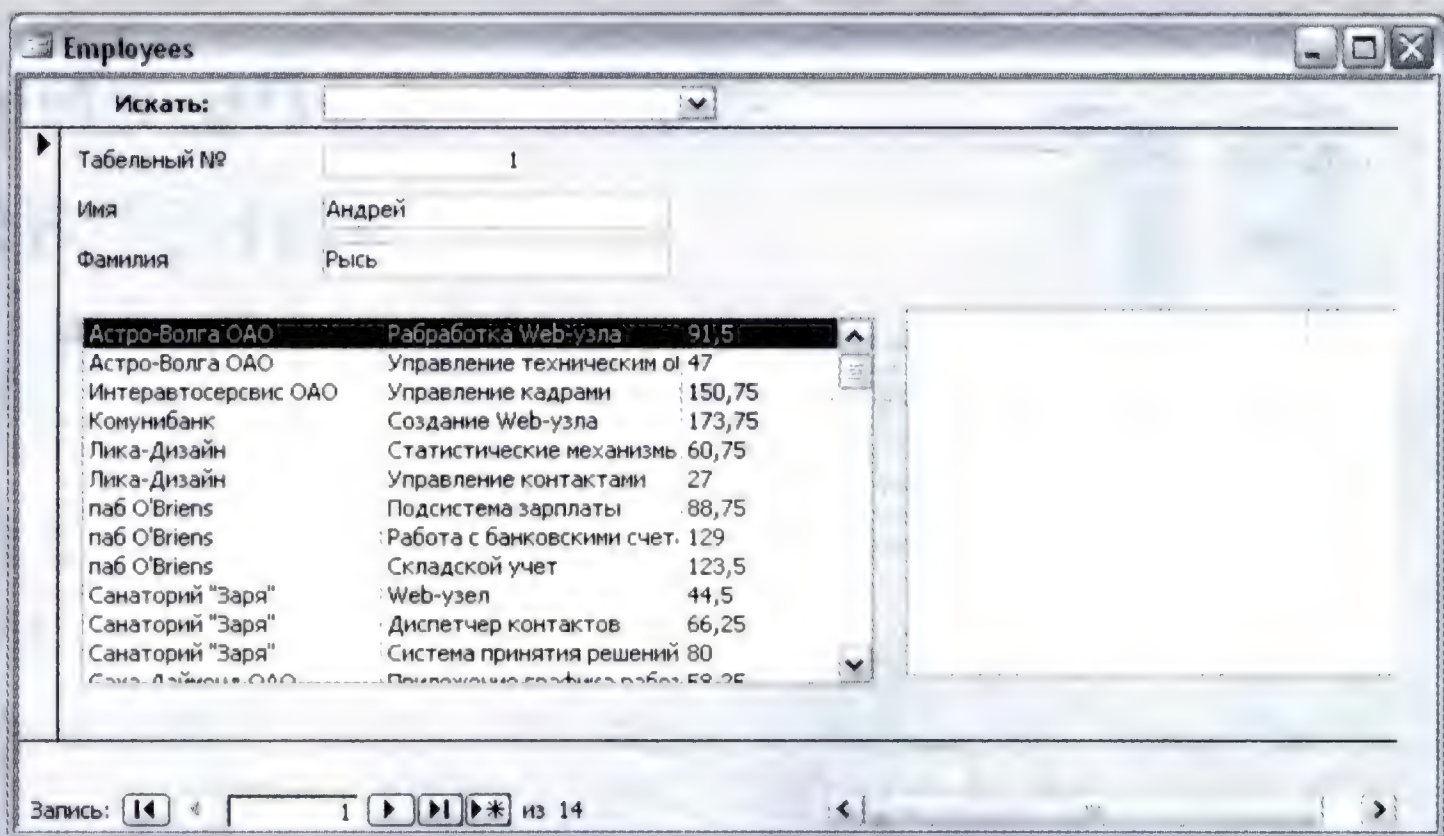


Рис. 12.21. С помощью списковых элементов можно больше узнать о проекте

Таблица 12.3. Свойства списковых элементов

Объект	Свойство	Значение
Список	Имя	lstProjects
	Источник строк	HoursWorkedByProject
	Число столбцов	4
	Ширина столбцов	0 см; 3,81 см; 3,81 см; 1,27 см
Список	Имя	lstDetails
	Число столбцов	4
	Ширина столбцов	0 см; 3,81 см; 1,27 см; 1,27 см

Переключите форму в режим формы. Первый сотрудник в списке – Андрей Рысь. Дважды щелкните на третьей позиции в списке проектов (задача Управление кадрами для Интеравтосервис ОАО, с общим объемом работ 150,75 часов). Теперь список задач во втором списковом элементе обновится (рис. 12.22).

Итак, подведем итоги: в течение девяти дней Андрей Рысь занимался разработкой в проекте задачи управления кадрами. Часть рабочего времени он затрачивал на тестирование, сопровождение и управление проектом. Каждая из задач имеет свою почасовую оплату; к тому же вы можете посмотреть, в какие именно дни выполнялась та или иная задача.

Несмотря на то, что описанный здесь подход не так часто встречается, как остальные, он может показаться вполне приемлемым и простым после подробного изучения списковых элементов, их свойств и особенностей поведения.

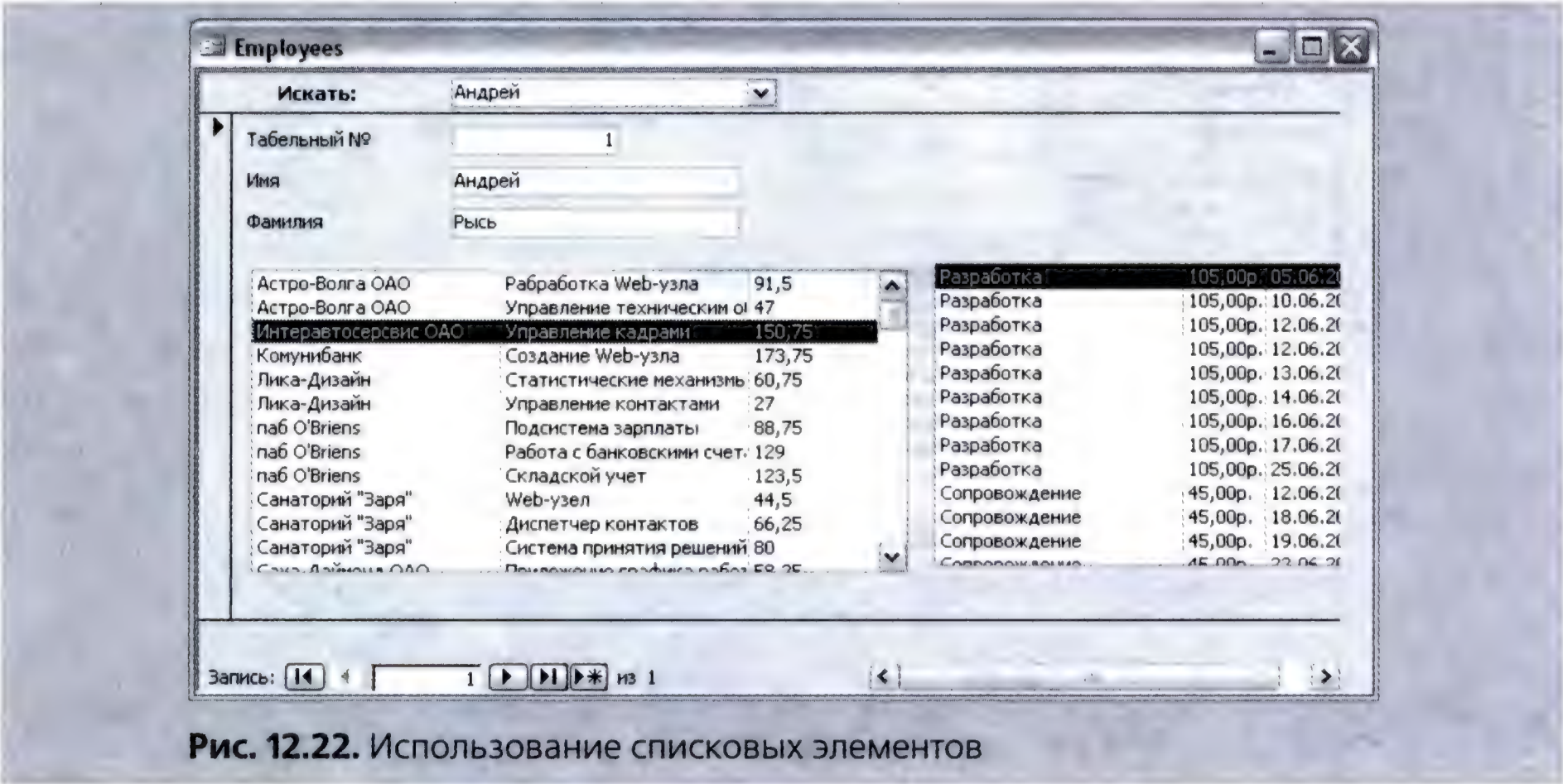


Рис. 12.22. Использование списковых элементов

Прочие элементы управления

13

Работа с текстовыми полями

Текстовые поля — вторые по популярности элементы управления, используемые в формах (наиболее часто используются метки, которые с точки зрения автоматизации нас не интересуют вообще). В практикуме главы 11 было продемонстрировано, как использовать события текстовых полей для предотвращения ввода пользователем недопустимых данных. Однако в VBA диапазоны возможностей текстовых полей гораздо шире. В этом разделе мы рассмотрим ряд ключевых свойств элементов текстовых полей, а после продемонстрируем ряд полезных методик работы с ними.

Ключевые свойства текстовых полей

Свойства текстовых полей отображаются в окне свойств, вызываемом в интерфейсе пользователя Access в режиме конструктора формы. С большинством из этих свойств можно работать непосредственно в тексте программы, однако некоторые из них используются в коде VBA чаще остальных. В табл. 13.1 перечислены наиболее широко используемые свойства текстовых полей.

В ЭТОЙ ГЛАВЕ

Работа с текстовыми полями.....	231
Использование элементов в группах переключателей	236
Работа с подформами.....	238
Работа со свойством Tag (Дополнительные сведения)	239

Совет

Не забывайте, что при в языке VBA обращении к объектам следует избегать пробелов в их именах. К примеру, объект текстового поля имеет свойство `ValidationText`, которое в окне свойств англоязычной версии Access имеет название `Validation Text`, а в русскоязычной – `Условие на значение`.

Таблица 13.1. Избранные свойства объекта текстового поля

Свойство объекта	Описание
<code>BackColor</code>	Цвет фона текстового поля
<code>BorderColor</code>	Цвет границы текстового поля
<code>ControlSource</code>	Поле в источнике данных, откуда текстовое поле получает данные
<code>Enabled</code>	<code>True</code> , если текстовое поле может получить фокус; <code>False</code> в противном случае
<code>FontBold</code>	<code>True</code> , если шрифт полужирный; <code>False</code> в противном случае
<code>FontItalic</code>	<code>True</code> , если шрифт курсивный; <code>False</code> в противном случае
<code>FontName</code>	Имя шрифта в текстовом поле
<code>FontSize</code>	Размер шрифта в текстовом поле
<code>ForeColor</code>	Цвет текста в поле
<code>Locked</code>	<code>True</code> , если в текстовое поле нельзя ввести данные; <code>False</code> в противном случае
<code>OldValue</code>	Исходное значение редактируемого текстового поля
<code>SelText</code>	Текст, выделенный в текстовом поле
<code>Tag</code>	Свойство, не используемое Access
<code>Text</code>	Текущий текст в текстовом поле
<code>Value</code>	Отредактированный текст в текстовом поле
<code>Visible</code>	<code>True</code> , если текстовое поле видимо, и <code>False</code> , — если нет

Эти свойства можно использовать для коррекции внешнего вида и поведения объекта во время работы пользователя с формой. Вы, наверняка, уже заметили, что в приведенном списке четыре свойства отвечают за текст, помещенный в поле. Среди этих свойств `SelText`, `OldValue` и `Text` доступны, только когда текстовое поле находится в фокусе. Если вы попытаетесь обратиться к какому-то из этих свойств, когда поле не будет находиться в фокусе, интерпретатор VBA выдаст сообщение об ошибке.

Еще одним свойством, требующим дополнительных пояснений, является свойство `Tag`. Его не использует Access — оно предоставляет дополнительное

место для хранения данных, ассоциированных с текстовым полем. Далее в этой главе будет описано, как это свойство можно использовать (см. раздел “Работа со свойством Tag (Дополнительные сведения)”).

Примечание

Свойство Tag имеют все формы, отчеты и элементы управления; его можно свободно использовать для хранения любых данных.

Отслеживание фокуса

Пользователи вашей базы данных могут иметь совершенно различный уровень подготовки. В то же время даже опытные пользователи могут не совсем четко себе представлять концепции, используемые вами, к примеру, концепции фокуса. Впервые столкнувшись с упоминанием о том, что только элементы в фокусе могут принимать данные, вы наверняка испытали замешательство. Представленная здесь методика поможет новичкам понять, какая часть формы в текущий момент является активной.

Для начала нам потребуется пара процедур в обособленном модуле. Одна из них устанавливает цвет фона (свойство BackColor) в желтый цвет, а вторая — в белый:

```
Sub HighlightControl(ctl As Control)
    ' Установка фона заданного элемента
    ' в желтый цвет
    On Error Resume Next
    ctl.BackColor = 65535
End Sub
```

```
Sub UnhighlightControl(ctl As Control)
    ' Установка фона заданного элемента
    ' в белый цвет
    On Error Resume Next
    ctl.BackColor = 16777215
End Sub
```

Как вы видите, эти процедуры имеют один аргумент типа Control. Control — это объект обобщенного типа, который можно использовать для представления любого элемента управления Access: текстового поля, комбинированного списка, надписи и т.п. Приложение Access позволит вам передать в эти процедуры любые объекты, не вызывая ошибки.

В то же время, нет никакой гарантии, что передаваемый вами элемент управления имеет свойство BackColor. Именно по этой причине в эти процедуры вставлена инструкция On Error Resume Next. Если в объекте свойство фона имеется, оно будет соответствующим образом изменено; если нет — произойдет безошибочный выход из процедуры.

Для использования этих процедур установите их вызов в функциях обработки событий получения и потери фокуса всех элементов формы. В качестве примера приведем совокупность таких процедур для формы Timeslips:


```

Private Sub DateWorked_GotFocus()
    HighlightControl DateWorked
End Sub

Private Sub DateWorked_LostFocus()
    UnhighlightControl DateWorked
End Sub

Private Sub EmployeeID_GotFocus()
    HighlightControl EmployeeID
End Sub

Private Sub EmployeeID_LostFocus()
    UnhighlightControl EmployeeID
End Sub

Private Sub Hours_GotFocus()
    HighlightControl Hours
End Sub

Private Sub Hours_LostFocus()
    UnhighlightControl Hours
End Sub

Private Sub TaskID_GotFocus()
    HighlightControl TaskID
End Sub

Private Sub TaskID_LostFocus()
    UnhighlightControl TaskID
End Sub

```

Сохраните модуль и вызовите данную форму. Вы увидите, что по мере передвижения между элементами в форме, фон элемента, находящегося в данный момент в фокусе, устанавливается в желтый цвет. На рис. 13.1 показана форма Timeslips после щелчка в поле Дата выполнения работ.

Рис. 13.1. Отслеживание фокуса открытой формы

Предупреждение

Если эту методику вы применяете в одной форме приложения, примените ее и во всех остальных, поскольку непоследовательные приложения очень сложно использовать.

Работа со свободными текстовыми полями

Свободным текстовым полем называют тот элемент, который не связан с каким-то конкретным полем таблицы базы данных. Свободные текстовые поля оказываются полезными для ввода пользователем переходных данных. К примеру, форма `BillingReportSetup13` базы данных `TimeTrack` разрешает пользователю выбрать клиента, даты начала и конца периода, после чего открыть отчет в режиме предварительного просмотра. В то же время данная форма не позволяет фильтровать результаты иначе как по дате. Вот как это организовать:

1. Откройте форму `BillingReportSetup13` в режиме конструктора.
2. Подведите указатель мыши под нижний край формы и перетяните ее границу вниз, освободив дополнительное место.
3. Добавьте в нижнюю часть формы текстовое поле. Поскольку сама форма свободна, текстовое поле по умолчанию тоже станет свободным. Назовите поле именем `txtWhere` и задайте его подпись как **Условие WHERE:**.
4. Добавьте в форму новую командную кнопку. Присвойте ей имя `cmdAdvancedReport` и метку **Расширенный отчет**.
5. Откройте модуль формы и добавьте в него следующую процедуру обработки нажатия кнопки расширенного отчета:

```
Private Sub cmdAdvancedReport_Click()
    On Error GoTo HandleErr
    DoCmd.OpenReport "BillingReport13", acViewPreview, _
        WhereCondition:=txtWhere.Value

ExitHere:
    Exit Sub
HandleErr:
    MsgBox "Ошибка "& Err.Number & ": " & _
        Err.Description & "в процедуре cmdAdvancedReport_Click"
    Resume ExitHere
End Sub
```

6. Теперь протестируйте форму.

Для тестирования новых элементов откройте форму в режиме формы. Выберите в качестве клиента **Комунибанк**, а в качестве дат начала и конца периода задайте, соответственно, **1/5/2004** и **1/7/2004**. В качестве Условия **WHERE** введите **Hours>10**. Теперь щелкните на кнопке **Расширенный отчет**. Как видно из рис. 13.2, в счет-фактуру попала только та работа, которая занимала более 10 часов.

Обратите внимание, что в приведенном фрагменте программы для получения информации из текстового поля, не находящегося в фокусе, использовано его свойство `Value`.

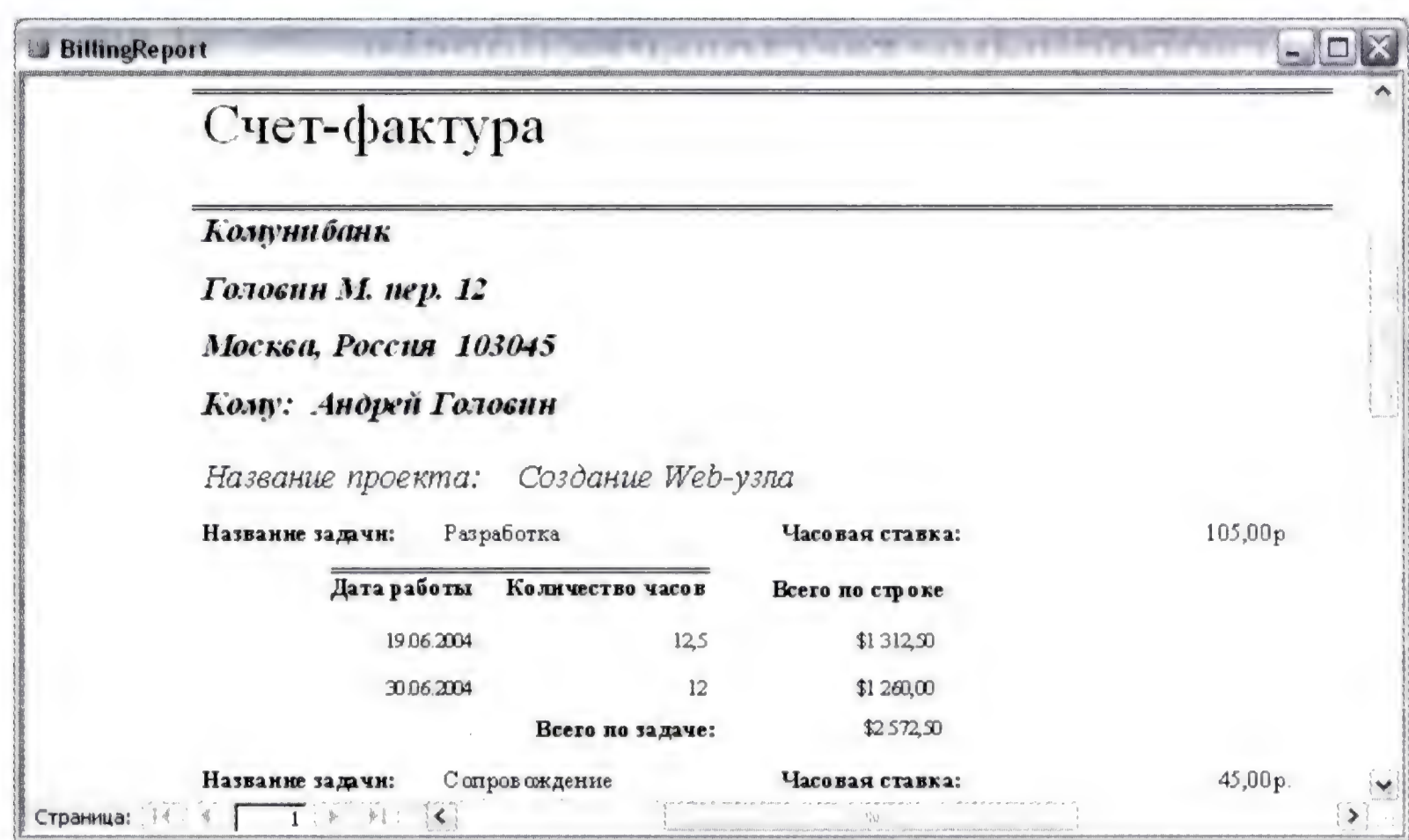


Рис. 13.2. Отчет, отфильтрованный во время выполнения

➔ Более подробно об использовании предложения WHERE в отчетах вы можете узнать в главе 14.

Использование элементов в группах переключателей

Группы переключателей представляют из себя элементы управления, которые сами могут содержать другие элементы. В частности, любые из следующих элементов можно поместить в группу переключателей, перетягивая их в режиме конструктора с панели элементов в область элемента группы:

- флажки;
- переключатели;
- выключатели.

В пределах одной группы может быть одновременно выбран только один элемент. К примеру, когда вы щелкаете на одном из переключателей группы, все остальные отключаются. Все элементы в группе переключателей имеют собственное свойство `OptionValue`; при этом свойство `Value` всей группы равно свойству `OptionValue` выбранного элемента.

Совет

Так как в пределах одной группы может быть выбран только один элемент, этот вариант группировки не подойдет, если вы захотите поместить на форму набор флажков, каждый из которых может устанавливаться независимо от других. В этом случае вы можете поместить отдельные флажки на форму, а для визуальной их группировки использовать прямоугольник.

Ни одно из полей базы данных TimeTrack не подходит для представления его группой переключателей, поэтому для демонстрации полезных процедур обработки событий была специально сконструирована форма Chapter13OptionGroup (рис. 13.3).

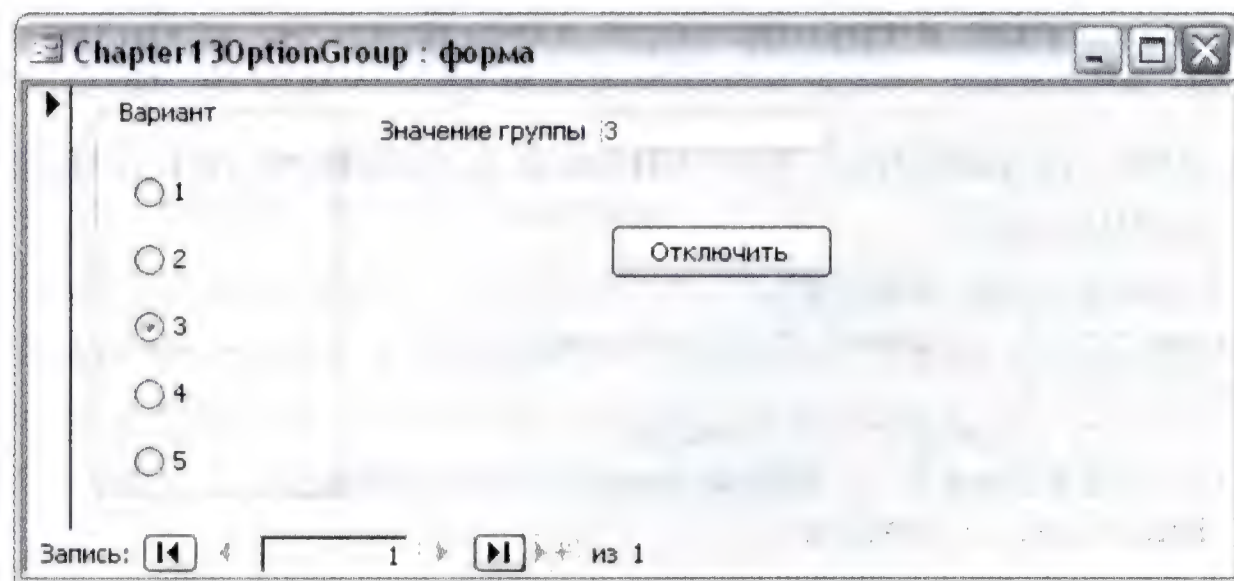


Рис. 13.3. Форма для тестирования процедур обработки событий группы переключателей

Демонстрационные процедуры, сопровождающие данную форму, создавались для выполнения трех задач:

- помещения значения группы переключателей в текстовое поле;
- установки значения группы переключателей при помощи ввода значения в текстовое поле;
- одновременного отключения всех кнопок в группе переключателей.

Для выполнения первой задачи нам потребуется написать обработку события После обновления (AfterUpdate) элемента группы переключателей, которое возникает при щелчке пользователем на одном из элементов группы:

```
Private Sub grpOption_AfterUpdate()  
    ' Показать значение группы переключателей  
    ' в текстовом поле  
    txtValue = grpOption  
End Sub
```

Единственная инструкция этой процедуры присваивает значение группы переключателей текстовому полю. Так как любой элемент имеет некоторое *свойство по умолчанию*, когда вы указываете имя объекта там, где должно по контексту находиться имя свойства, VBA использует именно его. Как в группе переключателей, так и в текстовом поле по умолчанию установлено свойство Value, вследствие чего упомянутая инструкция устанавливает значение одного элемента в значение другого.

Для выполнения второй задачи потребуется установить обработку события Изменение (Change) элемента текстового поля, которое возникает при вводе пользователем какого-либо символа.

```
Private Sub txtValue_Change()  
    ' Обновление группы переключателей  
    ' из текстового поля
```



```
On Error Resume Next
grpOption = CInt(txtValue.Text)
End Sub
```

Инструкция `On Error Resume Next` в данной процедуре следит за тем, чтобы пользователь не ввел в текстовое поле ошибочные данные, например символ `q`. В этом случае в группе переключателей не производится никаких изменений.

И наконец, для отключения всех кнопок переключателей было введено несколько новых концепций:

```
Private Sub cmdDisable_Click()
' Отключение всей группы переключателей
Dim ctl As Control
For Each ctl In grpOption.Controls
If ctl.ControlType = acOptionButton Then
ctl.Enabled = False
End If
Next ctl
End Sub
```

Элементы, которые содержат в себе другие элементы (подобно группе переключателей), имеют собственные коллекции `Controls`. Используя цикл `For...Each`, мы в процедуре проходим по всем элементам коллекции при щелчке пользователя на кнопке `Отключить`. В то же время, если вы присмотритесь внимательно к форме на рис. 13.3, то увидите, что в группе переключателей содержатся не только элементы переключателей, но и элементы надписей. Если попытаться отключить элемент надписи, произойдет ошибка выполнения. Вместо написания обработчика ошибок в этой процедуре был предпринят более элегантный подход. Все элементы форм и отчетов Access имеют свойство `ControlType`, которое содержит константу данного типа элемента. Поэтому, проходя по всем элементам группы, мы проверяем, не является ли данный элемент переключателем, и только в случае положительного результата проверки мы отключаем элемент.

Работа с подформами

Подформы (или подчиненные формы) — это особый тип элементов управления. Они сами в себе содержат формы, которые, в свою очередь, содержат собственные элементы, в том числе другие подформы (правда, в Access глубже подформ 1-го уровня проникнуть невозможно).

Для работы с подформами прежде всего нужно знать, как обращаться к элементам, которые они содержат. В общем случае для обращения к элементам подформ используется следующий синтаксис:

```
Forms! [имя_главной_формы] . [имя_элемента_подформы] . Form! [имя_элемента]
```

В качестве примера возьмем форму `Projects` из базы данных `TimeTrack`, которая содержит элемент подформы с именем `Tasks`. Чтобы отобразить значе-

ние поля `TaskName` текущей строки подформы, в окне `Immediate` нужно набрать следующую строку:

```
?Forms!Projects.Tasks.Form!TaskName
```

Естественно, если программа работает в пределах формы `Projects`, вы можете сократить эту ссылку за счет использования ключевого слова `Me`. Вот пример присвоения в таком случае предыдущего значения некоторой переменной:

```
varTaskName = Me.Tasks.Form!TaskName
```

Изучив способ обращения к элементам подформ, вы должны знать, что работа с ними производится в точности так же, как и с обычными формами. Ключевым здесь является свойство `Form`. Именно оно позволяет “заглянуть вовнутрь” подформы и обратиться к содержащимся в ней элементам.

Работа со свойством Tag (Дополнительные сведения)

Начинающих программистов VBA иногда смущает свойство дополнительных сведений `Tag`: зачем нужно было в конструкцию объекта вводить свойство, которое никогда не используется в Access? На самом деле причина заключается в том, что разработчики приложения заранее предусмотрели для программистов своеобразную “нишу”, которую они могут использовать по своему усмотрению. Все элементы управления Access имеют свойство `Tag`, и уже только вам решать, что в него помещать и как его использовать.

Чтобы показать пример использования свойства `Tag`, попробуем видоизменить процедуру `HighlightControl`, с которой вы уже встречались в этой главе. На данный момент эта процедура при переходе фокуса к некоторому элементу устанавливает его фон в желтый цвет. А как изменить этот цвет на другой?

В Access не существует отдельного свойства элементов, позволяющего хранить цвет для выделения фона элементов. В данном случае следует использовать свойство `Tag`. Итак внесем в процедуру изменения:

1. Откройте форму `Timeslips` в режиме конструктора. Установите свойство **Дополнительные сведения** элементов `EmployeeID` и `TaskID` в значение `65535`. Теперь установите это же свойство элементов `DateWorked` и `Hours` в значение `16777088`.
2. Откройте программный модуль `Chapter13` и модифицируйте процедуру `HighlightControl` следующим образом:

```
Sub HighlightControl(ct1 As Control)
    ' Установка фона заданного элемента
    ' в желтый цвет
    On Error Resume Next
    If Not IsNull(ct1.Tag) Then
        ct1.BackColor = ct1.Tag
    Else
```



```
        ctl.BackColor = 65535  
    End If  
End Sub
```

Теперь сохраните форму и переключитесь в режим формы. Пройдите с помощью клавиши табуляции по всем полям формы. Вы видите, что при переходе фокуса в верхние два элемента цвет подсветки изменяется на желтый (что соответствует коду фона 65535), а при переходе к следующим двум элементам подсветка становится светло-голубой (что соответствует коду фона 16777088).

Предупреждение

Не путайте свойство **Дополнительные сведения (Tag)** со свойством **Смарт-теги (SmartTags)**.

Обратите внимание, что процедура всегда выполняет какое-то действие независимо от того, установлено или нет в элементах свойство **Дополнительные сведения**. Если данное свойство установлено, то при перемещении к этому полю цвет фона изменяется на заданный в свойстве; в противном случае — на желтый, как и в первой версии программы.

ПРАКТИКУМ

Создание главной формы просмотра

В практикуме этой главы мы создадим альтернативный интерфейс для отображения информации о клиентах, проектах и сотрудниках. Идея состоит в том, чтобы обеспечить пользователя некоей главной формой, которая позволит ему переключаться между этими тремя типами информации. При вводе данных вы можете применить подсветку элементов.

Для начала нужно создать три формы, которые будут выступать подформами главной. Создадим их с размерами три дюйма ширины на два дюйма высоты и присвоим им имена `ClientSub`, `EmployeeSub` и `ProjectSub`. Свойства дополнительных сведений текстовых полей этих форм мы установим в значение 65535, а комбинированного списка формы `ProjectSub` — в значение 12615935, чтобы дать понять пользователю, что здесь значения берутся из другой таблицы. На рис. 13.4 показаны эти три формы, открытые в режиме конструктора.

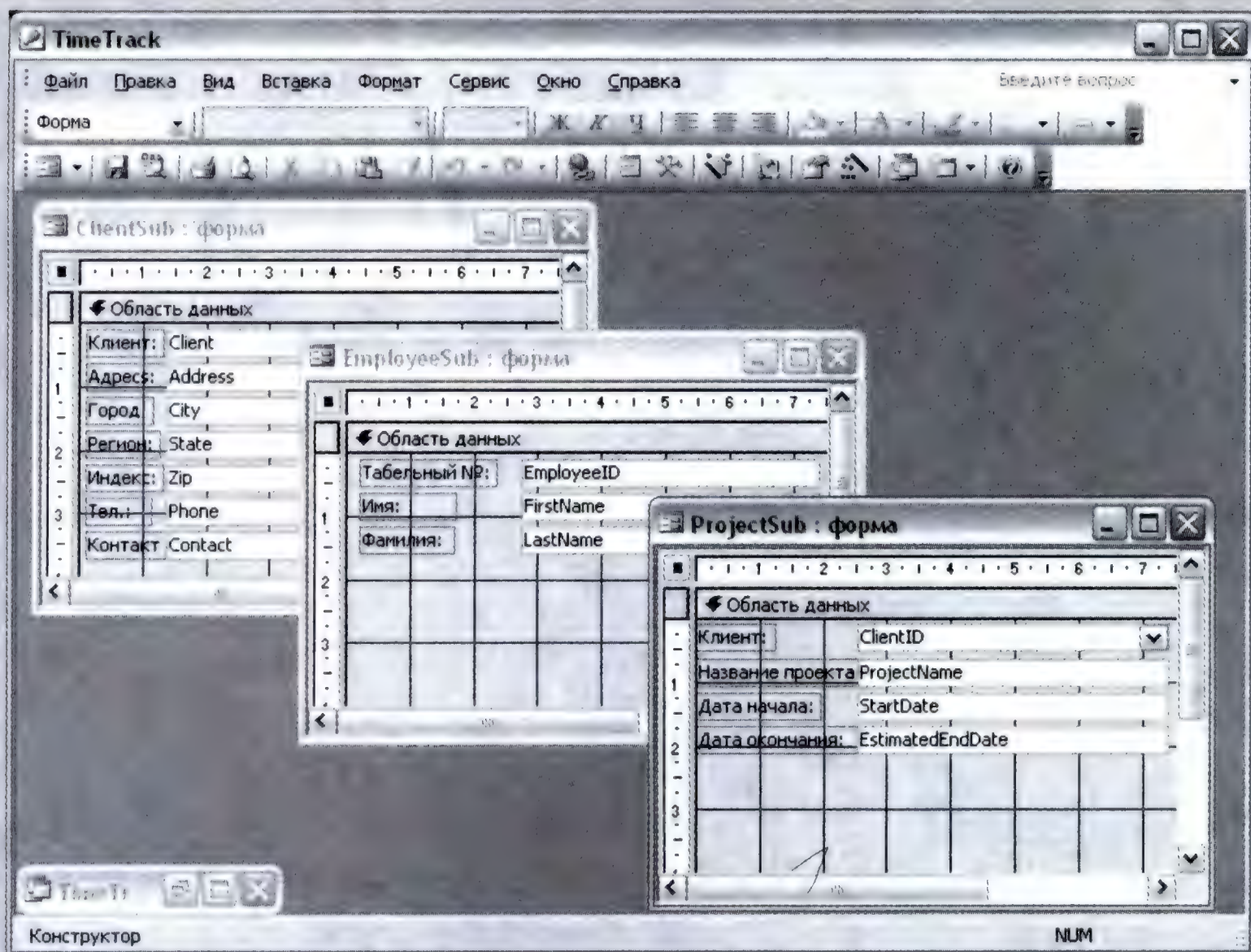


Рис. 13.4. Подформы альтернативного интерфейса пользователя

Естественно, каждая из форм имеет свои программы для изменения фона элементов. Содержимое программного модуля формы ClientSub следующее:

```
Option Compare Database
Option Explicit
```

```
Private Sub Address_GotFocus()
    HighlightControl Address
End Sub
```

```
Private Sub Address_LostFocus()
    UnhighlightControl Address
End Sub
```

```
Private Sub City_GotFocus()
    HighlightControl City
End Sub
```

```
Private Sub City_LostFocus()
    UnhighlightControl City
End Sub
```

```
Private Sub Client_GotFocus()
    HighlightControl Client
End Sub
```

```
Private Sub Client_LostFocus()
    UnhighlightControl Client
End Sub
```



```
Private Sub Contact_GotFocus()
    HighlightControl Contact
End Sub
```

```
Private Sub Contact_LostFocus()
    UnhighlightControl Contact
End Sub
```

```
Private Sub Phone_GotFocus()
    HighlightControl Phone
End Sub
```

```
Private Sub Phone_LostFocus()
    UnhighlightControl Phone
End Sub
```

```
Private Sub State_GotFocus()
    HighlightControl State
End Sub
```

```
Private Sub State_LostFocus()
    UnhighlightControl State
End Sub
```

```
Private Sub Zip_GotFocus()
    HighlightControl Zip
End Sub
```

```
Private Sub Zip_LostFocus()
    UnhighlightControl Zip
End Sub
```

А вот соответствующий модуль формы EmployeeSub:

```
Option Compare Database
Option Explicit
```

```
Private Sub EmployeeID_GotFocus()
    HighlightControl EmployeeID
End Sub
```

```
Private Sub EmployeeID_LostFocus()
    UnhighlightControl EmployeeID
End Sub
```

```
Private Sub FirstName_GotFocus()
    HighlightControl FirstName
End Sub
```

```
Private Sub FirstName_LostFocus()
    UnhighlightControl FirstName
End Sub
```

```
Private Sub LastName_GotFocus()
    HighlightControl LastName
End Sub
```

```
Private Sub LastName_LostFocus()
```



```
UnhighlightControl LastName
End Sub
```

И в заключение, программный модуль формы ProjectSub:

```
Option Compare Database
Option Explicit
```

```
Private Sub ClientID_GotFocus()
    HighlightControl ClientID
End Sub
```

```
Private Sub ClientID_LostFocus()
    UnhighlightControl ClientID
End Sub
```

```
Private Sub EstimatedEndDate_GotFocus()
    HighlightControl EstimatedEndDate
End Sub
```

```
Private Sub EstimatedEndDate_LostFocus()
    UnhighlightControl EstimatedEndDate
End Sub
```

```
Private Sub ProjectName_GotFocus()
    HighlightControl ProjectName
End Sub
```

```
Private Sub ProjectName_LostFocus()
    UnhighlightControl ProjectName
End Sub
```

```
Private Sub StartDate_GotFocus()
    HighlightControl StartDate
End Sub
```

```
Private Sub StartDate_LostFocus()
    UnhighlightControl StartDate
End Sub
```

Следующим шагом будет создание главной формы. Эта форма является свободной и содержит три подформы. Выполните следующие действия:

1. Создайте новую форму в режиме конструктора.
2. Добавьте в форму группу переключателей и присвойте ей имя grpSub.
3. Добавьте в группу три кнопки. Установите их подписи в следующие значения: Клиенты, Сотрудники и Проекты, а соответствующие свойства **Значение параметра** – в 1, 2 и 3.
4. Перетяните форму ClientSub из окна базы данных на главную форму. Это – простейший способ автоматической привязки размеров подформы к размерам главной формы.
5. Удалите метку, которую программа Access создала для подформы.
6. Назовите элемент подформы именем SwitchForm. На рис. 13.5 показана созданная главная форма с подформой в режиме конструктора.

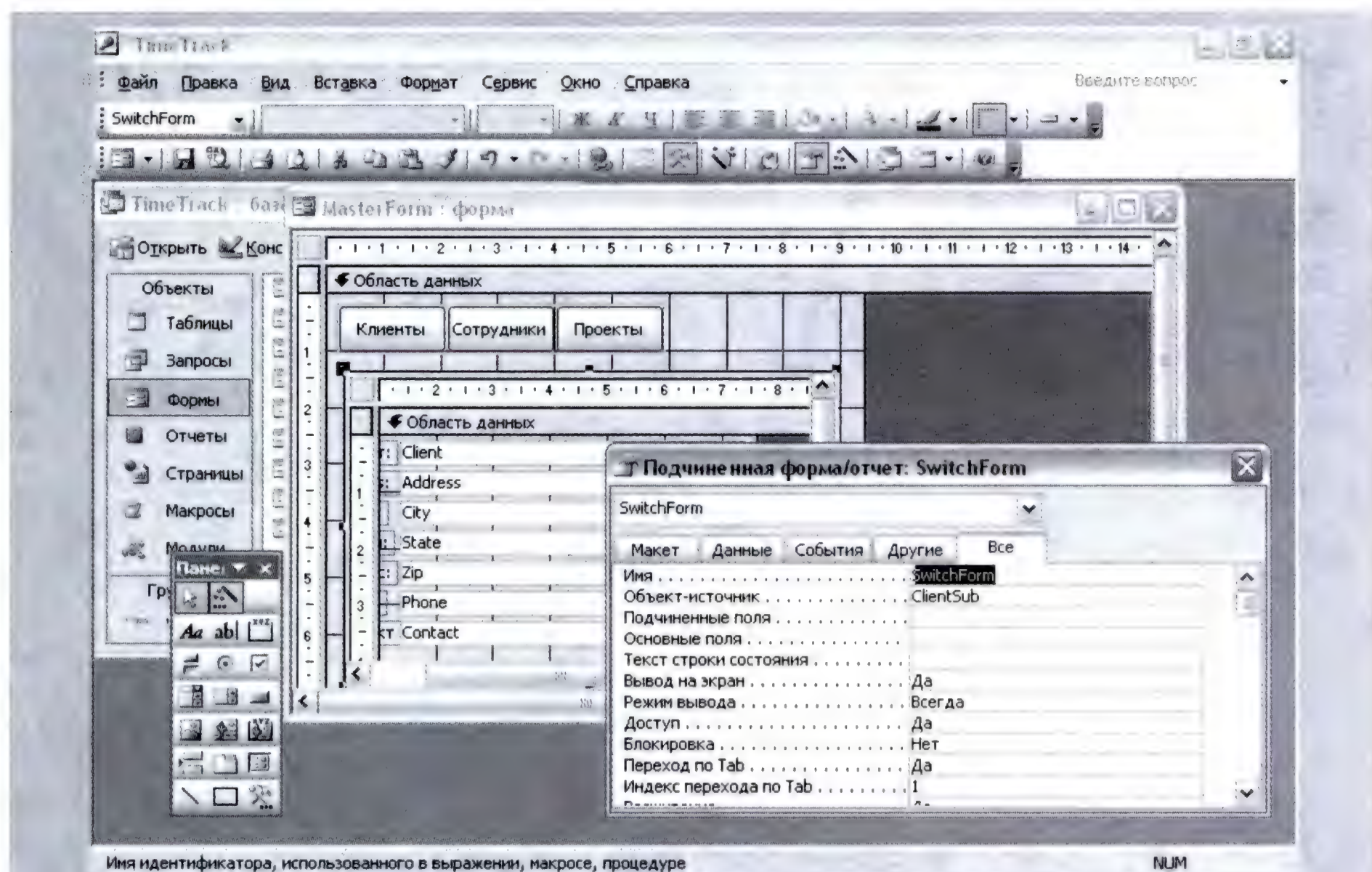


Рис. 13.5. Создание главной формы

7. Добавьте в модуль формы процедуру, переключающую содержимое подформы:

```
Private Sub grpSub_AfterUpdate()  
    ' Переключение подформ при нажатии кнопок  
    Select Case grpSub.Value  
        Case 1 ' Клиенты  
            Me!SwitchForm.SourceObject = "ClientSub"  
        Case 2 ' Сотрудники  
            Me!SwitchForm.SourceObject = "EmployeeSub"  
        Case 3 ' Проекты  
            Me!SwitchForm.SourceObject = "ProjectSub"  
    End Select  
End Sub
```

Сохраните форму и откройте ее в представлении формы. Вы обнаружите, что при нажатии кнопок главной формы содержимое подформ соответствующим образом изменяется, а при переходе между полями изменяется фон активного элемента. На рис. 13.6 показан заключительный вариант главной формы.

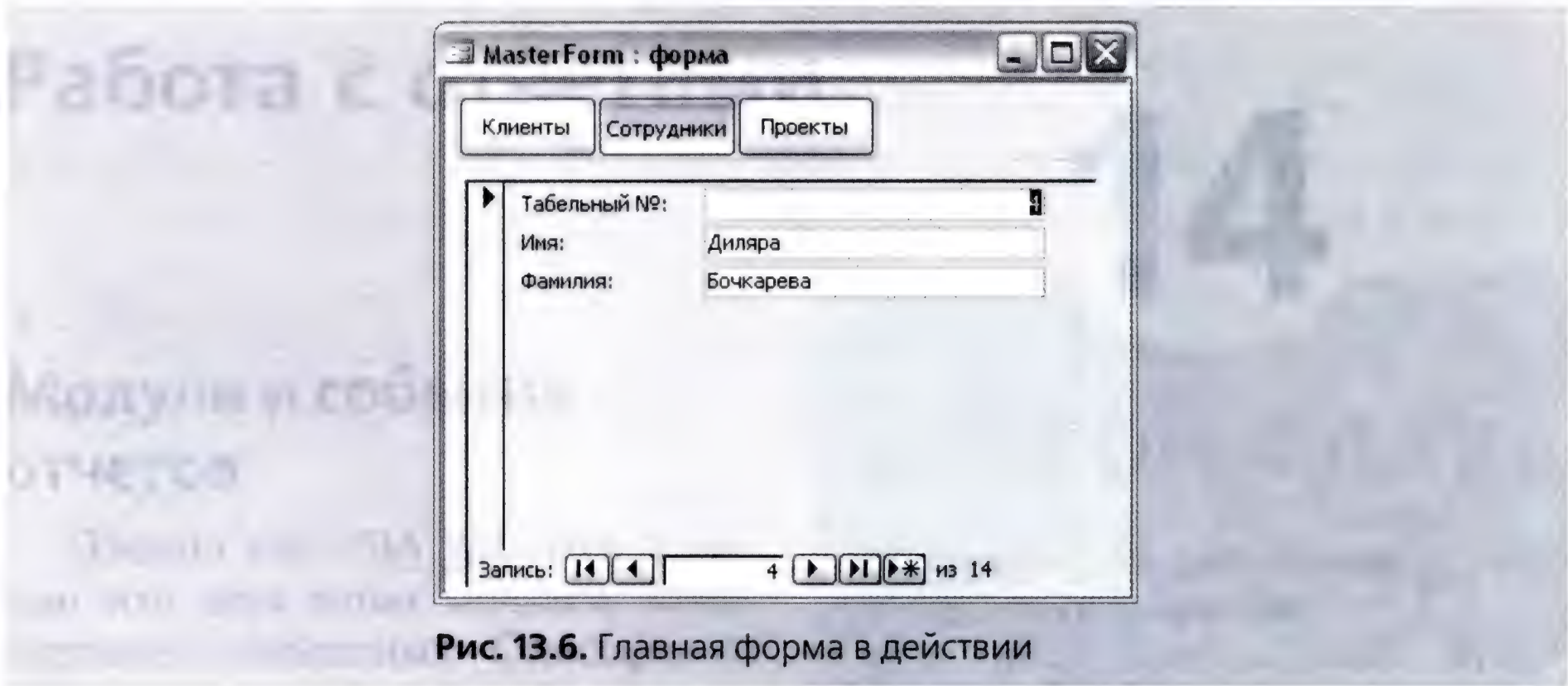


Рис. 13.6. Главная форма в действии

Работа с отчетами

14

Модули и события отчетов

Обычно код VBA хранится в одном или двух типах модулей: стандартных и объектных. Стандартные модули являются обособленными, объектные же прикреплены к формам и отчетам. Второй тип модулей сохраняется вместе с объектами и поддерживает события и свойства, специфичные для данных объектов.

Большая часть программ, которая вводится и сохраняется в модулях отчетов и форм, состоит из процедур обработки событий — программ, которые вызываются при выполнении пользователем каких-либо действий с формой или отчетом. Например, процедуры, обрабатывающие событие Открытие (Open) отчета, выполняются при открытии отчета. Рассмотрим простой пример, в котором отчет разворачивается на экране.

1. Выделите отчет `BillingReport13` в окне базы данных и щелкните на кнопке **Программа** панели инструментов. Откроется редактор VBE, в котором активным будет окно модуля текущего отчета.
2. Введите следующий фрагмент программы:

В ЭТОЙ ГЛАВЕ

Модули и события отчетов	247
Открытие и закрытие отчетов.....	248
Передача данных в отчет через аргумент <code>OpenArgs</code>	250
Наполнение отчета данными...	252
Обработка ошибок уровня отчета	254
Использование VBA для определения свойств группировки в отчете	257


```
Private Sub Report_Open(Cancel As Integer)
    DoCmd.Maximize
End Sub
```

3. Щелкните на кнопке Save (Сохранить).
 4. Вернитесь в окно База данных Access и закройте отчет.
 5. Откройте форму BillingReportSetup13. Эта форма создана для запуска отчета BillingReport после того, как пользователь введет несколько значений, ограничивающих число записей, на основе которых будет создан отчет.
 6. Выберите клиента Астро-Волга ОАО и введите даты 1/10/04 и 31/10/04.
 7. Щелкните на кнопке Открыть отчет. В результате этого действия будет вызвано событие открытия отчета (Open), запущена процедура его обработки, и окно предварительного просмотра отчета развернется на экране.
- ➔ События отчетов и их последовательность подробно рассмотрены в разделе “Последовательность событий в отчетах” главы 11.

Открытие и закрытие отчетов

Одним из способов, с помощью которых можно провести пользователя по рабочему процессу, является открытие для него объектов, с которыми ему предстоит работа, и закрытие их по завершению этой работы. Естественно, большинство пользователей и сами способны открывать и закрывать объекты, однако автоматизация этих действий поможет не сбиться пользователю с пути.

Открытие отчетов

В практикуме главы 8 мы преобразовали макрос, открывающий отчет BillingReport13, в следующую процедуру обработки события:

```
Private Sub cmdOpenReport_Click()
    On Error GoTo HandleErr
    DoCmd.OpenReport "BillingReport13", acViewPreview
ExitHere:
    Exit Sub
HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description & "в процедуре cmdBillingReport13_Click"
    Resume ExitHere
End Sub
```

В настоящий момент нас интересует метод OpenReport, вызываемый в этой процедуре. Он очень похож на метод OpenForm, о котором мы говорили в главе 10. Оба метода принадлежат объекту DoCmd. Для открытия отчета используют метод OpenReport, имеющий следующий синтаксис:


```
DoCmd.OpenReport имя_отчета [, вид] [, фильтр] [, условие] _  
[, режим_окна] [, операнды]
```

где *имя_отчета* — строковое выражение, идентифицирующее отчет, открываемый по имени. В табл. 14.1 перечислены необязательные аргументы этого метода; в табл. 14.2 и 14.3 приводятся дополнительные сведения о его синтаксисе.

Таблица 14.1. Необязательные аргументы метода OpenReport

Аргумент	Тип данных	Описание
<i>вид</i>	Constant	Одна из встроенных констант, перечисленных в табл. 14.2. Определяет представление отчета
<i>фильтр</i>	Variant	Строковое выражение, равное имени существующего запроса
<i>условие</i>	Variant	Строковое выражение, равное допустимому предложению WHERE языка SQL (WHERE является ключевым словом)
<i>режим_окна</i>	Constant	Одна из встроенных констант, перечисленных в табл. 14.3. Определяет режим открытия отчета
<i>операнды</i>	Variant	Строковое выражение или значение, передаваемое свойству отчета OpenArgs

Таблица 14.2. Константы аргумента *вид*

Константа	Значение	Описание
acViewNormal	0	Вывод отчета на печать
acViewDesign	1	Открытие отчета в режиме конструктора
acViewPreview	2	Открытие отчета в режиме предварительного просмотра

Таблица 14.3. Константы аргумента *режим_окна*

Константа	Значение	Описание
acWindowsNormal	0	Режим, заложенный в свойствах отчета
acHidden	1	Окно отчета открывается и скрывается
acIcon	2	Окно отчета открывается и сворачивается в значок на панели задач Windows
acDialog	3	Отчет открывается как модальное диалоговое окно (свойства Всплывающее окно (Modal) и Модальное окно (Popup) установлены в Да (Yes))

Метод `OpenReport`, который вызывался в примере главы 8, имел следующий вид:

```
DoCmd.OpenReport "BillingReport13", acViewPreview
```

и открывал отчет `BillingReport13` в окне предварительного просмотра. Несмотря на существование множества аргументов, в этом случае для выполнения работы потребовалось всего два из них.

Сам отчет основывается на запросе `BillingReportSource13`, возвращающем значения, используемые в отчете. Источник данных можно передавать в аргументе *фильтр*; в то же время можно построить предложение `WHERE` и использовать аргумент *условие* для ограничения диапазона записей, выводимых в отчет. Часто существует множество способов решения одной и той же задачи; при этом обычно из них выбирают самый подходящий для конкретных условий.

Заккрытие отчета

Для того чтобы закрыть отчет, можно воспользоваться методом `Close` объекта `DoCmd`, имеющим следующий синтаксис:

```
DoCmd.Close [тип_объекта] [, объект] [, сохранить]
```

В этой конструкции все аргументы необязательны. Когда все аргументы при вызове опущены, закрывается объект, находящийся в настоящий момент в фокусе. Если вы обращаетесь к объекту по имени, используйте аргумент *объект*, а если по типу — укажите в аргументе *тип_объекта* константу `acReport`. В качестве аргумента *сохранить* можно использовать три константы:

- `acSavePrompt` (значение 0). Открывает окно, запрашивающее у пользователя подтверждение сохранения.
- `acSaveYes` (значение 1). Безоговорочно сохраняет изменения в отчете.
- `acSaveNo` (значение 2). Изменения не сохраняются.

Передача данных в отчет через аргумент `OpenArgs`

Вы уже знаете, как передавать значения в процедуру с помощью аргументов. Аналогично значения можно передавать и в отчеты, однако на этот раз следует использовать не обычную структуру процедуры, а аргумент и свойство `OpenArgs`.

Аргумент `OpenArgs` принадлежит методу `OpenReport`; именно посредством его организуется передача значений в отчет. Свойство же принадлежит отчету — в него принимаются и в нем сохраняются переданные через метод `OpenReport` значения.

➔ Использование метода и свойства `OpenArgs` в формах описано в соответствующем разделе главы 10.

Предположим, что в отчете счета-фактуры не всегда нужно видеть подробности, иногда достаточно только сводных строк. Используя аргумент `Open`

Args, можно передать в отчет значение, которое при необходимости блокирует вывод подробных сведений. Чтобы добавить такую функциональность в базу данных TimeTrack, выполним следующее:

1. Откроем форму BillingReportSetup13 в режиме конструктора и добавим в нее элемент флажка, разместив его прямо под кнопкой открытия отчета (здесь имеется в виду кнопка Открыть отчет, а не кнопка Расширенный отчет, созданная в главе 13). Присвоим имени элемента значение chkSummary, а подписи — Показывать подробности.

2. Установите значение по умолчанию данного элемента в –1. В результате по умолчанию флажок будет установлен.

3. Измените процедуру обработки события нажатия кнопки, модифицировав в ней вызов метода OpenReport следующим образом:

```
DoCmd.OpenReport "BillingReport13",  
acViewPreview,,,str(chkSummary)
```

При этом аргумент OpenArgs передаст свойству OpenArgs формы значение флажка chkSummary (установлен он или снят), предварительно преобразовав его в строковый тип.

4. Сохраните и закройте форму.

5. В процедуру обработки события Open отчета BillingReport13 сразу после инструкции DoCmd.Maximize добавьте следующую строку:

```
Reports("BillingReport13").Section(acDetail).Visible =  
Me.OpenArgs
```

6. Сохраните и закройте отчет.

7. Вернитесь в измененную форму. Выберите клиента Астро-Волга ОАО и введите даты 01/10/04 и 31/10/04.

8. Снимите флажок Показывать подробности (рис. 14.1). После этого щелкните на кнопке Открыть отчет. На рис. 14.2 показан отчет, содержащий только обобщенные значения и не имеющий строк детализированной информации. Закройте форму и отчет.

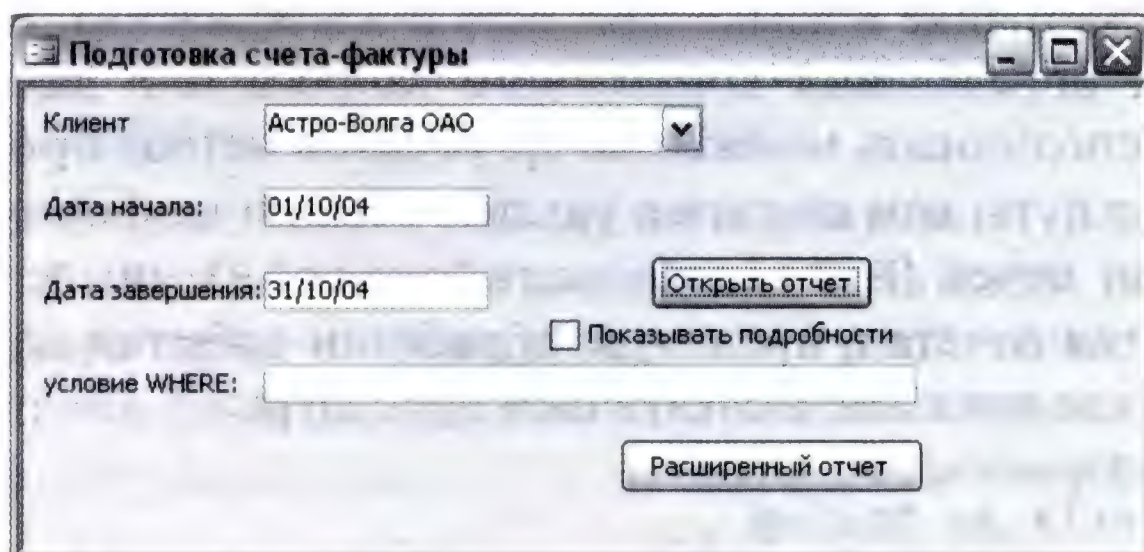


Рис. 14.1. Снимите флажок в форме настройки отчета

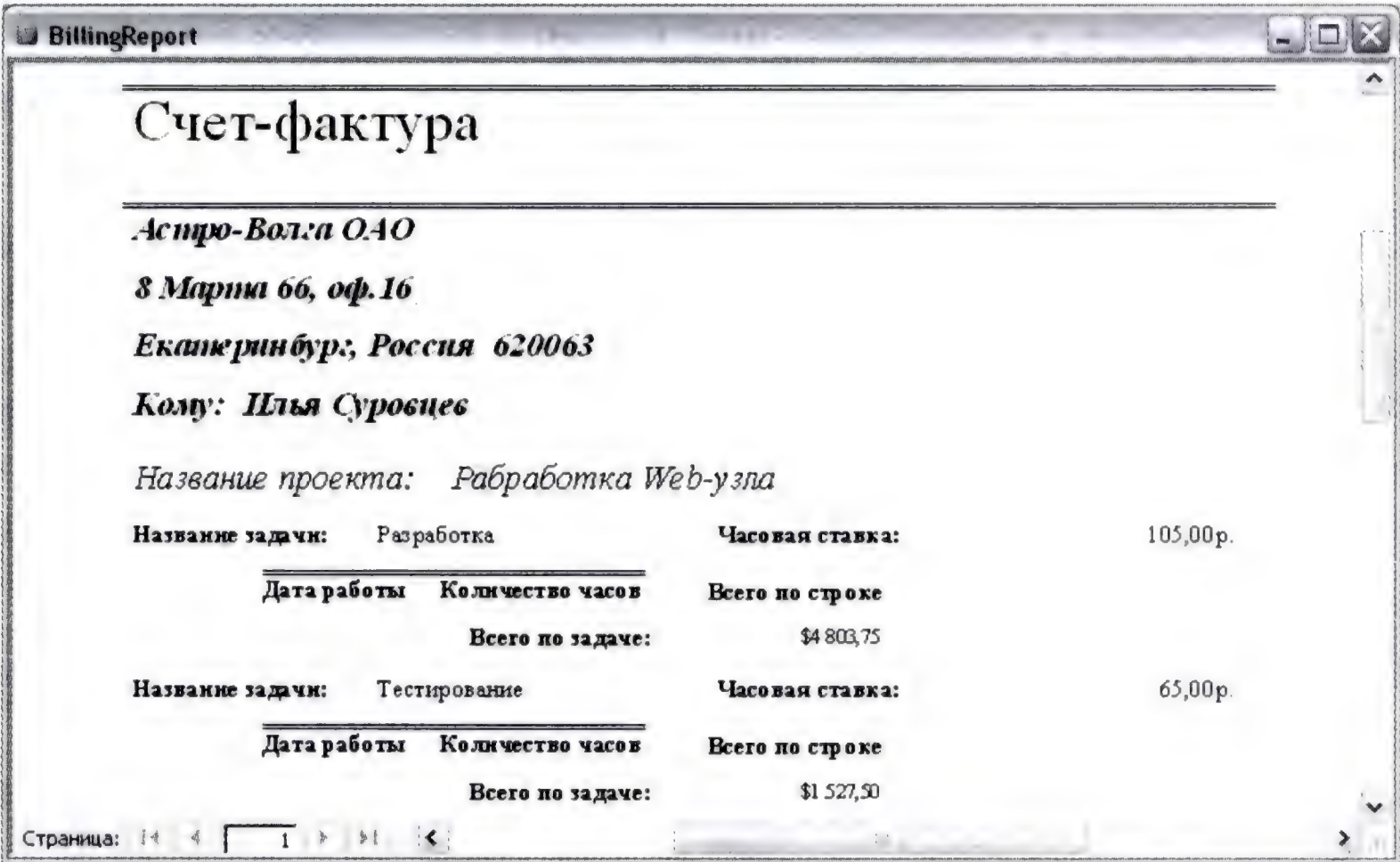


Рис. 14.2. Здесь аргумент `OpenArgs` использован для переключения между детализированным и сводным отчетом

Если установить флажок (что соответствует значению по умолчанию), в отчет будет передано значение `True` или `-1`. Если этот флажок снять, отчету будет передано значение `False` или `0`. В последнем случае свойству `Visible` (Отображать) раздела подробностей отчета также присвоится значение `False` (Ложь), и этот раздел не будет отображен в отчете.

Наполнение отчета данными

Существующий отчет счета-фактуры (`BillingReport13`) демонстрирует один из способов ограничения множества записей, формирующих отчет. Запрос, на котором основан отчет, получает значения из формы настройки отчета (`BillingReportSetup13`). Эта методика достаточно проста и не требует написания большего числа программных инструкций, нежели короткая процедура вызова отчета из формы.

Однако этой методике в языке VBA есть и альтернатива. Как уже говорилось ранее, для определения конкретного запроса или даже предложения `WHERE` можно использовать множество аргументов метода `OpenReport`. Если вы выберете этот путь, вам придется удалить ссылки на форму из запроса, на котором основан отчет (`BillingReportSource13`), после чего заместить команды открытия отчета в процедуре обработки события нажатием кнопки **Открыть отчет**, как показано в следующей процедуре:

```
Private Sub cmdOpenReport_Click()  
    Dim strCriteria As String  
    Dim frm As Form  
    Set frm = Forms!BillingReportSetup13  
    On Error Goto HandleErr
```



```

strCriteria = "Clients.ClientID = " & frm!cboClient.Value & _
"AND Timeslips.DateWorked Between #" & frm!txtStartDate & _
"# AND #" & frm!txtEndDate & "#"
Debug.Print strCriteria
DoCmd.OpenReport "BillingReport13", acViewPreview, , _
str(strCriteria)

ExitHere:
Exit Sub
HandleErr:
MsgBox "Ошибка " & Err.Number & ": " & _
Err.Description
Resume ExitHere
End Sub

```

Этот метод оказался более сложным, так как в данном случае пришлось формировать предложение WHERE. В то же время не существует определенных правил выбора решения задачи. Со временем вы поймете, что каждый из методов имеет свои плюсы и минусы, а выбор следует производить исходя из требований конкретной задачи.

Применение фильтра и сортировки

На данный момент вы уже видели достаточное количество строковых выражений и констант, используемых для ограничения объема записей, которые выводятся в отчет счета-фактуры базы данных примеров TimeTrack. Однако существует еще одно свойство фильтрации, с которым вы еще не сталкивались. Как формы, так и отчеты имеют свойства, позволяющие активизировать применение фильтров к рассматриваемому набору записей. Перечислим эти свойства:

- **Фильтр (Filter).** Строковое выражение, идентифицирующее текущий фильтр. Оно подобно предложению WHERE без ключевого слова WHERE.
- **Фильтр включен (FilterOn).** Включает фильтр, определенный в свойстве Фильтр.
- **Порядок сортировки (OrderBy).** Определяет порядок сортировки записей, однако естественный порядок сортировки в отчете имеет преимущество над данным свойством.
- **Сортировка включена (OrderByOn).** Включает сортировку, определенную в свойстве Порядок сортировки.

Установка свойств Filter и FilterOn в программе VBA является одним из способов автоматизации этого процесса, позволяющим достаточно гибко использовать один и тот же отчет в применении к разным наборам данных.

Установка строки фильтрации имеет следующий синтаксис:

```
Me.Filter = строка_фильтрации
```

где *строка_фильтрации* представляет собой критерий, согласно которому данные отчета фильтруются. Например, если вы хотите отфильтровать счет-

фактуру по конкретному клиенту, то можете использовать следующую инструкцию:

```
Me.Filter = "ClientID = 1"
```

В некоторых случаях может потребоваться вслед за этой инструкцией активизировать свойство `FilterOn`:

```
Me.FilterOn = True
```

Установка свойства `FilterOn` в значение `False` отключает применение текущего фильтра.

Свойства сортировки аналогичны свойствам фильтрации. Они неявно задают применение предложение `ORDER BY` языка SQL. В качестве примера установки порядка сортировки приведем следующие инструкции:

```
Me.OrderBy = "ClientID, ProjectID DESC"
Me.OrderByOn = True
```

Обработка ошибок уровня отчета

Как и в формах, вы можете обрабатывать специфичные для отчетов ошибки, используя специальное событие `Error`. Это событие инициируется механизмом Jet, когда при его работе возникает ошибка. Данное событие имеет два аргумента, передающие событию следующие значения:

- `DataErr`. В этом аргументе сохраняется код ошибки, возвращенный объектом `Err`.
- `Response`. Этот аргумент определяет, будет ли отображаться сообщение об ошибке. Он может принимать значения двух констант: `acDataErrContinue`, при которой ошибка игнорируется, и `acDataErrDisplay`, при которой пользователю выводится стандартное системное сообщение об ошибке.

Основное достоинство события `Error` состоит в том, что вы можете найти список цифровых кодов ошибок и выбрать из него те, в которых стандартная обработка вас не устраивает. Эти коды вы можете вынести в собственную процедуру обработки события `Error` и выполнить для них те действия, которые считаете нужным. Рассмотрим простой пример на основе отчета `BillingReport`.

1. Откройте отчет `BillingReport13` в режиме конструктора и измените его источник записей с `BillingReportSource13` на `test`.
2. Сохраните отчет и закройте его.
3. Теперь откройте форму `BillingReportSetup13`, выберите клиента и щелкните на кнопке **Открыть отчет**. При открытии окна предварительного просмотра отчета будет сгенерирована ошибка, показанная на рис. 14.3.

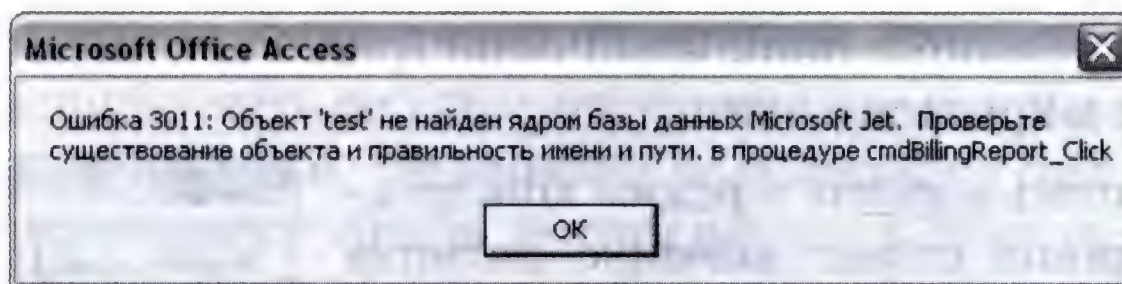


Рис. 14.3. Замена источника данных отчета привела к ошибке механизма Jet

4. Щелкните на кнопке ОК и закройте окно сообщения.
5. Откройте модуль отчета и введите следующую процедуру обработки события Error:


```
Private Sub Report_Error(DataErr As Integer, Response As Integer)
    Debug.Print DataErr
End Sub
```
6. Теперь снова откройте отчет из окна баз данных в режиме предварительного просмотра, чтобы ошибка была сгенерирована.
7. Переключитесь в окно Immediate редактора VBE — в нем будет отображен код ошибки (рис. 14.4).

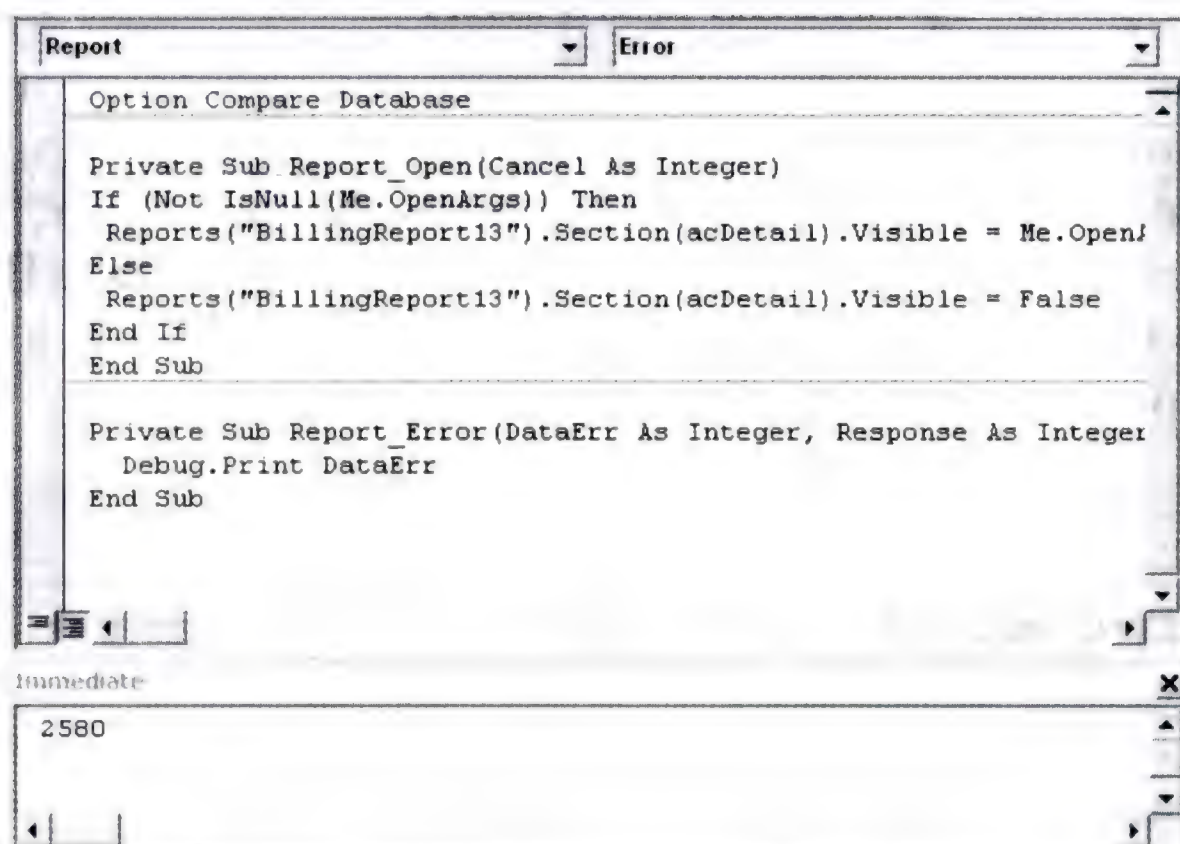


Рис. 14.4. Код ошибки Jet можно вывести в окно Immediate

8. Теперь вы знаете код ошибки и можете заменить инструкцию `Debug.Print` фрагментом реальной обработки ошибки:


```
Private Sub Report_Error(DataErr As Integer, Response As Integer)
    If DataErr = 2580 Then
        MsgBox "Источник данных отчета изменен или " & _
            "отсутствует", vbOKOnly, "Ошибка"
        Response = acDataErrContinue
    End If
End Sub
```
9. Еще раз откройте отчет в режиме предварительного просмотра из окна базы данных. Возникнет та же ошибка, но на этот раз будет отображено

уже запрограммированное вами окно (рис. 14.5). Щелкните на кнопке ОК, чтобы закрыть это окно.

- 10. Переключитесь в отчете в режим конструктора и верните старое значение свойства Данные — BillingReportSource13.

В данном конкретном случае мы добавили в обработку события Error только собственное сообщение об ошибке. Оно не имеет приоритета перед системными сообщениями об ошибке, вызываемыми методом DoCmd.OpenReport, так что увидеть его возможно только при вызове отчета из окна базы данных. При вызове отчета из формы подготовки данных BillingReportSetup13 генерируется сообщение об ошибке, предусмотренное в процедуре обработки события кнопки Click. Однако в приведенном примере нас интересовал только один конкретный момент — способ вставки собственной обработки ошибки отчета, и мы добились своей цели.

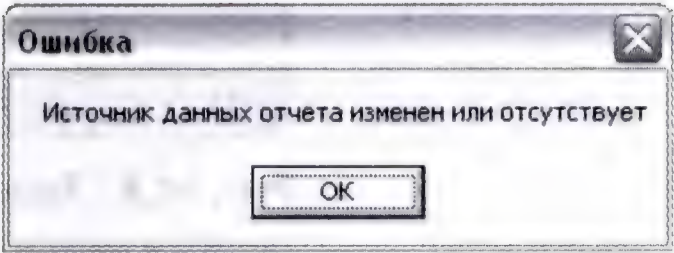


Рис. 14.5. За счет обработки события Error можно отображать собственное сообщение

Что делать при отсутствии данных?

Отчеты имеют одну свойственную только им проблему: что возвращать, когда в источнике данных нет удовлетворяющих критерию записей? На самом деле это совсем не ошибка, и Access все равно выводит отчет, однако в большинстве случаев пользователю трудно понять, что он видит. В качестве примера на рис. 14.6 показан отчет BillingReport, открытый из окна баз данных, а не из формы подготовки данных.

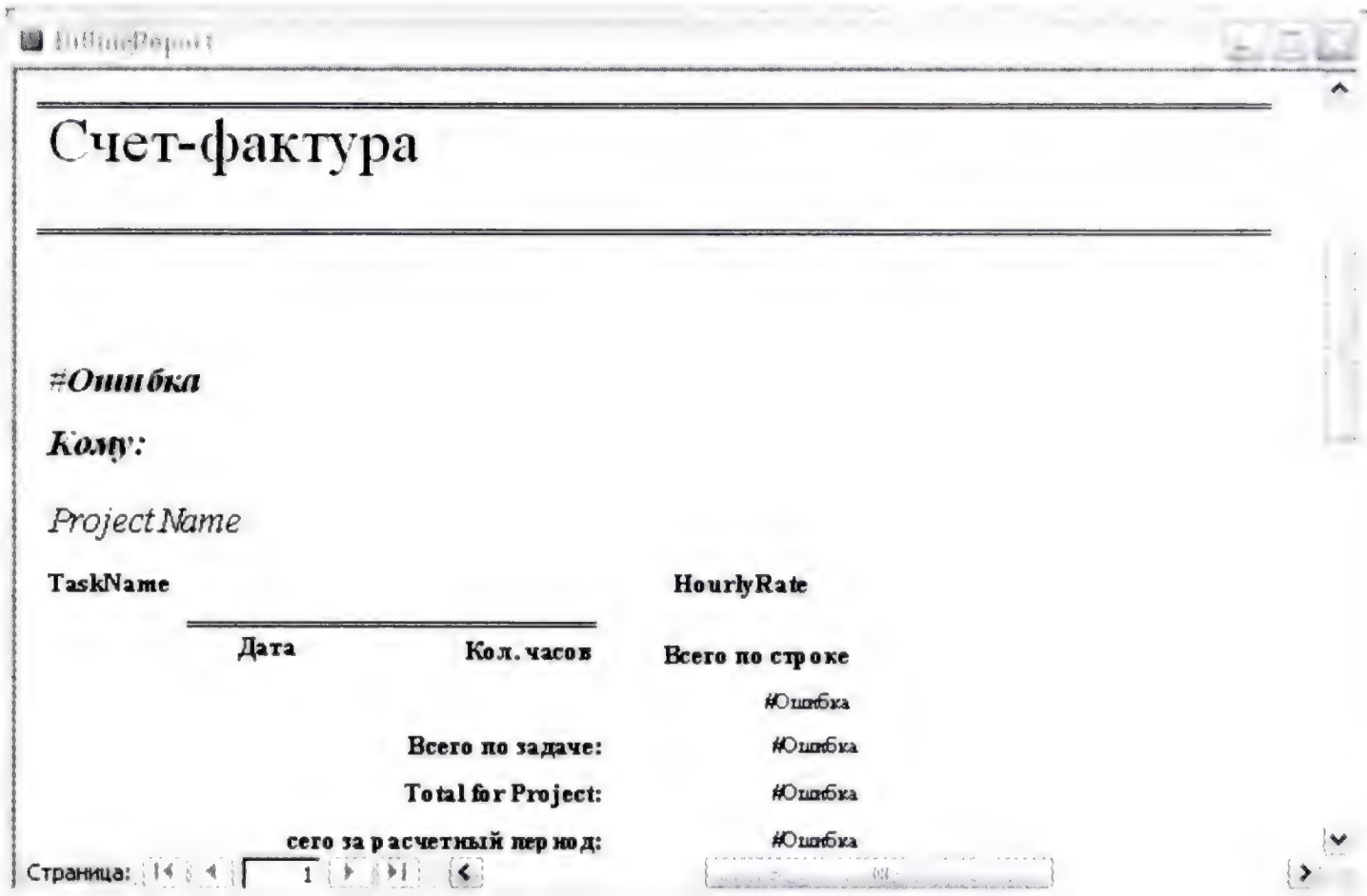


Рис. 14.6. Отчет, не содержащий значений, выводится наполненным сообщениями об ошибках

Данный отчет полон сообщениями об ошибках. Чтобы избежать этой проблемы, можно использовать событие отсутствия данных и создать его собственную процедуру обработки, где вывести пользователю соответствующее сообщение и прекратить вывод отчета. В этом разделе в качестве примера мы будем использовать отчет `BillingReport14` и проиллюстрируем простейшее решение проблемы. Для начала откройте модуль отчета `BillingReport14` и введите в него следующую процедуру:

```
Private Sub Report_NoData(Cancel As Integer)
    MsgBox "Нет записей для отображения", _
        vbOKOnly, "Отчет остановлен"
    Cancel = True
End Sub
```

Переключитесь в режим формы и на все запросы параметров нажимайте клавишу `<Enter>`. Эти запросы возникают по той причине, что Access пытается разрешить ссылки на отсутствующие значения. После того как Access не получит на свои запросы данные, откроется сообщение, заложенное нами в процедуре обработки события отсутствия данных (рис. 14.7). Щелкните на кнопке `ОК` и закройте окно сообщения.

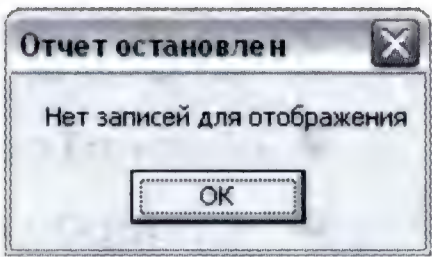


Рис. 14.7. Сообщите пользователю об отсутствии данных для отчета

Использование VBA для определения свойств группировки в отчете

Колонтитулы групп имеют соответствующие события, равно как и другие разделы отчета, и эти события можно использовать для изменения отчета “на лету” (при этом можно даже добавлять новые разделы).

Ссылка на компоненты отчета в программе

При работе с разделами и группами отчета нужно знать, как ссылаться на соответствующие области или уровни. Для ссылки на разделы отчета используются их имена или номера. К примеру, для ссылки на раздел **Область данных** можно использовать следующий синтаксис:

```
Reports(имя_отчета).Detail
```

где *именем_отчета* является строка, идентифицирующая отчет по имени. `Detail` — это фактическое имя раздела области данных. Для обращения можно также использовать следующий синтаксис:

```
Reports(имя_отчета).Section(0)
```

или даже

```
Reports(имя_отчета).Section(acDetail)
```

Для ссылок на уровни группировки используется та же форма:

```
Reports(имя_отчета).GroupLevel(индекс).свойство
```

Уровень группировки является на самом деле объектом, так что его можно использовать в качестве переменной, например:


```
Dim glGroup As GroupLevel
Set glGroup = Reports("BilingReports").GroupLevel(0)
```

Предупреждение

При ссылке на разделы отчета и уровни группировки всегда нужно явным образом указывать свойство, которое читается или записывается. Ни разделы отчета, ни уровни группировки не имеют свойств по умолчанию.

Объект GroupLevel имеет следующие свойства, с которыми следует познакомиться будущему специалисту в области построения отчетов в Access:

- GroupFooter (Примечание группы);
- GroupHeader (Заголовок группы);
- GroupInterval (Интервал);
- GroupOn (Группировка);
- KeepTogether (Не разрывать);
- SortOrder (Порядок сортировки);
- ControlSource (Источник группировки).

В табл. 14.4 перечислены допустимые значения каждого из этих свойств.

Таблица 14.4. Значения свойств группировки

Свойство	Установка	Числовое значение
GroupHeader GroupFooter	Имеется	True
	Не имеется	False
GroupOn	По полному значению	0
	По первым символам	1
	По годам	2
	По кварталам	3
	По месяцам	4
	По неделям	5
	По дням	6
	По часам	7
	По минутам	8
	По интервалу	9
KeepTogether	Нет	0
	Всю группу	1
	С первой строкой данных	2
SortOrder	По возрастанию	False
	По убыванию	True

Предупреждение

Новые группы можно создавать только в режиме конструктора, однако большую часть свойств можно установить в процедуре обработки события отчета `Open`.

ПРАКТИКУМ**Составление дневного отчета**

В базе данных примеров `TimeTrack` содержится всего один отчет, и он имеет дело со счетами-фактурами. Предположим, что вашим разработчикам потребовалось увидеть исправленный график работ. Для представления такой информации больше всего подойдет отчет, поэтому пользователям нужно предоставить простой способ вывода графика работ, сгруппированного по дням, по неделям и даже по месяцам. Это требует создания формы, с помощью которой пользователь сможет выбрать один из трех видов графика работ. Для этого больше подойдет главная кнопочная форма `SwitchBoard`.

1. Откройте форму `Switchboard` в режиме конструктора и вставьте в нее три командные кнопки. Присвойте им следующие имена (и метки): `cmdDaily` (График: день), `cmdWeekly` (График: неделя) и `cmdMonthly` (График: месяц).
2. В модуле формы введите следующие процедуры обработки событий:

```
Private Sub cmdDaily_Click()  
    Call GenerateSchedule(6)  
End Sub
```

```
Private Sub cmdMonthly_Click()  
    Call GenerateSchedule(4)  
End Sub
```

```
Private Sub cmdWeekly_Click()  
    Call GenerateSchedule(5)  
End Sub
```

```
Sub GenerateSchedule(rpt As String)  
    DoCmd.OpenReport "Schedule", acViewPreview, , , , rpt  
End Sub
```

При каждом вызове процедуры `GenerateSchedule` ей передается значение, представляющее значение свойства `GroupOn` группировки (см. табл. 14.4). Метод `OpenReport` передает это значение отчету с помощью аргумента `OpenArgs`. Сам отчет пока еще не существует, но в этом нет ничего страшного.

3. Сохраните и закройте форму `Switchboard`, которая теперь приняла вид, показанный на рис. 14.8.
4. Создайте новый запрос с именем `Schedule`, показанный на рис. 14.9.
5. Сохраните и закройте запрос.
6. Используя Мастер табличных автоотчетов, создайте отчет, основываясь на только что созданном запросе `Schedule`. Присвойте ему имя `Schedule`.

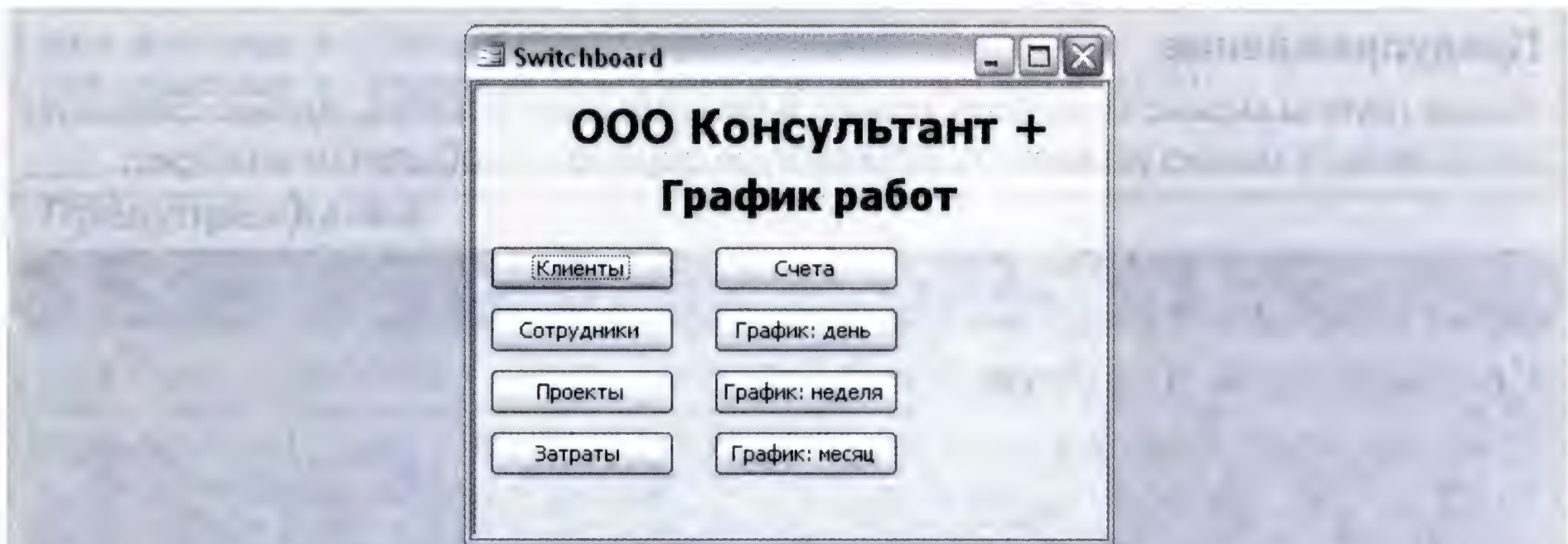


Рис. 14.8. Добавьте на кнопочную панель три новые командные кнопки

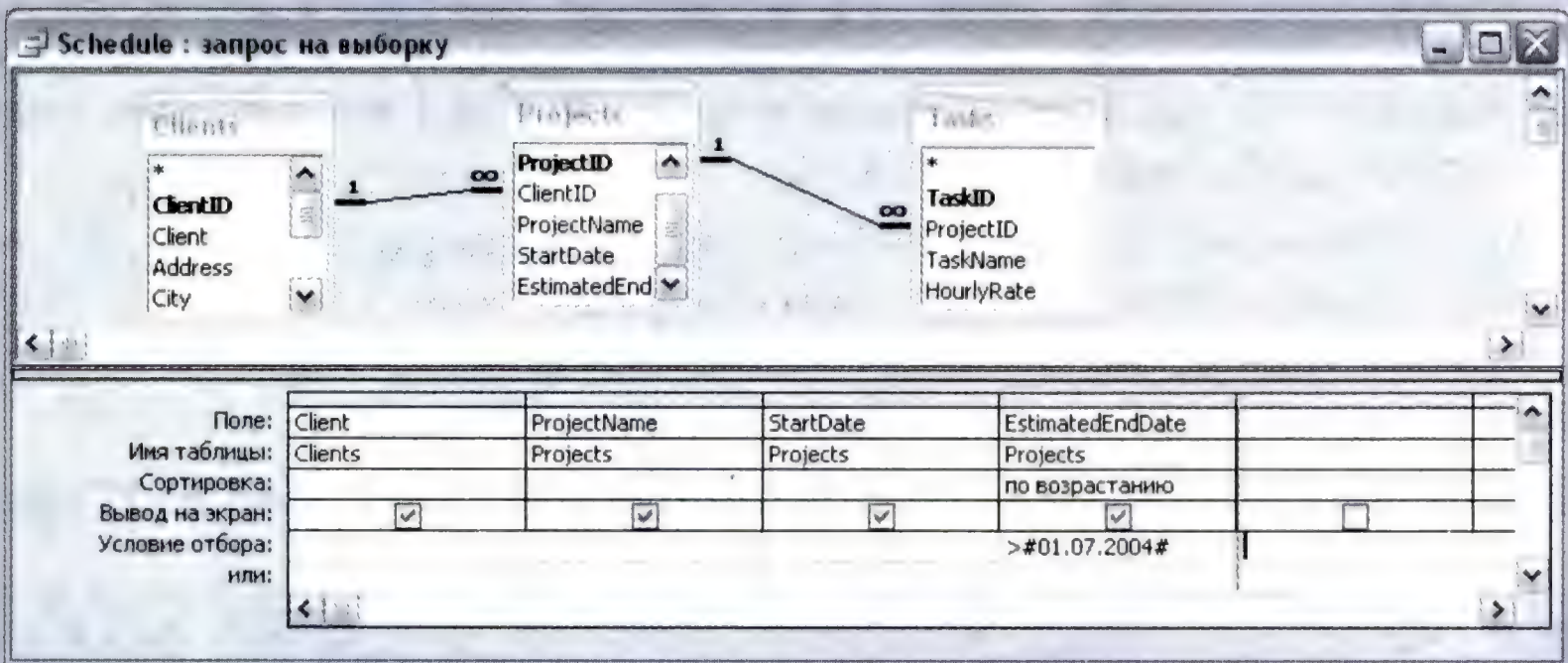


Рис. 14.9. Отчет графика работ будет базироваться на этом запросе

7. Откройте новый отчет в режиме конструктора и щелкните на инструменте Сортировка и группировка. Используя рис. 14.10 в качестве руководства, установите свойства групп. Группировка по каждому из дней будет принята в отчете по умолчанию. Свойства первой и третьей строк оставьте как есть. Свойство наличия заголовка во второй строке установите в значение Да, что поможет вам определять, где закончилась одна группа и начинается следующая.

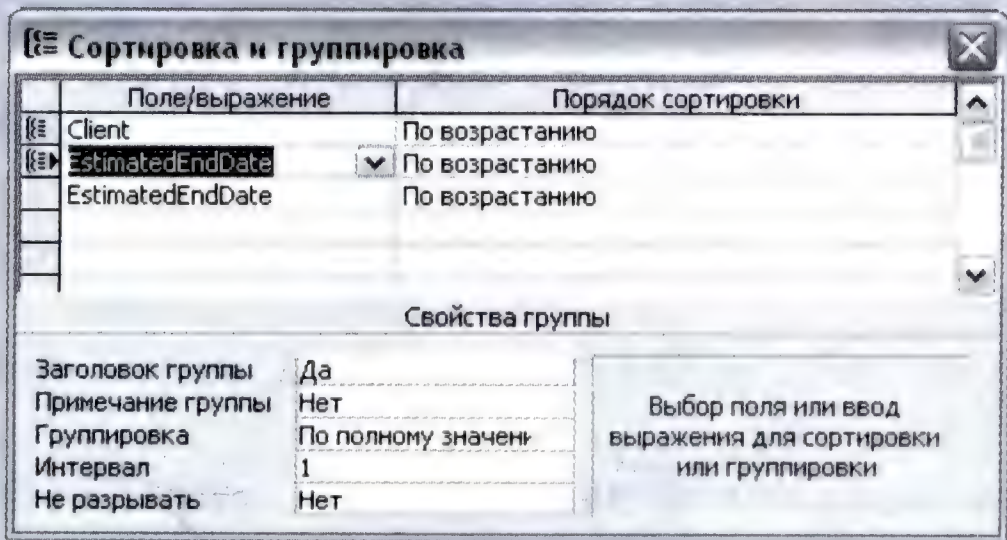


Рис. 14.10. Установка начальных свойств группировки

8. В модуле отчета введите следующую процедуру:


```
Private Sub Report_Open(Cancel As Integer)
    If IsNull(Me.OpenArgs) Then
        Exit Sub
    End If
    Me.GroupLevel(1).GroupOn = CInt(Me.OpenArgs)
End Sub
```

9. Сохраните и закройте отчет.

Откройте форму переключателей и щелкните на новой кнопке **График: дни**, чтобы просмотреть график, сгруппированный по полю `EstimatedEndDate` (на рис. 14.11 представлена только показательная часть отчета). Этот отчет сгруппирован по каждому из значений (каждый день создает новую группу), что равноценно умолчанию, которое было использовано ранее. В этом случае нет никакой необходимости передавать отчету константу дня, как мы сделали в примере, однако это совсем не повредит, особенно если впоследствии вы будете модифицировать первый уровень группировки, который в настоящее время базируется на поле `Client`.

Управление списком рассылок	24.02.2004	29.12.2004
Управление списком рассылок	24.02.2004	29.12.2004
Сопровождение расчетов	26.05.2004	05.03.2005
Сопровождение расчетов	26.05.2004	05.03.2005
Сопровождение расчетов	26.05.2004	05.03.2005
Сопровождение расчетов	26.05.2004	05.03.2005
Зарплата	08.03.2004	27.03.2005
Зарплата	08.03.2004	27.03.2005
Зарплата	08.03.2004	27.03.2005
Зарплата	08.03.2004	27.03.2005
Диспетчер склада	04.06.2004	16.04.2005
Диспетчер склада	04.06.2004	16.04.2005
Диспетчер склада	04.06.2004	16.04.2005

Рис. 14.11. В этом отчете группировка проводится по полю даты окончания `EstimatedEndDate`

Закройте отчет и щелкните на кнопке **График: месяц**. В отчете, показанном на рис. 14.12, отображаются те же данные, что и в первом, однако группы значений поля `EstimatedEndDay` сформированы по месяцам. Как видите, в двух отчетах по-разному сгруппированы работы в марте 2005 года. Из-за достаточной разницы в датах окончания работ в базе данных примеров понедельный отчет ничем не будет отличаться от ежедневного, но это совсем не показатель.

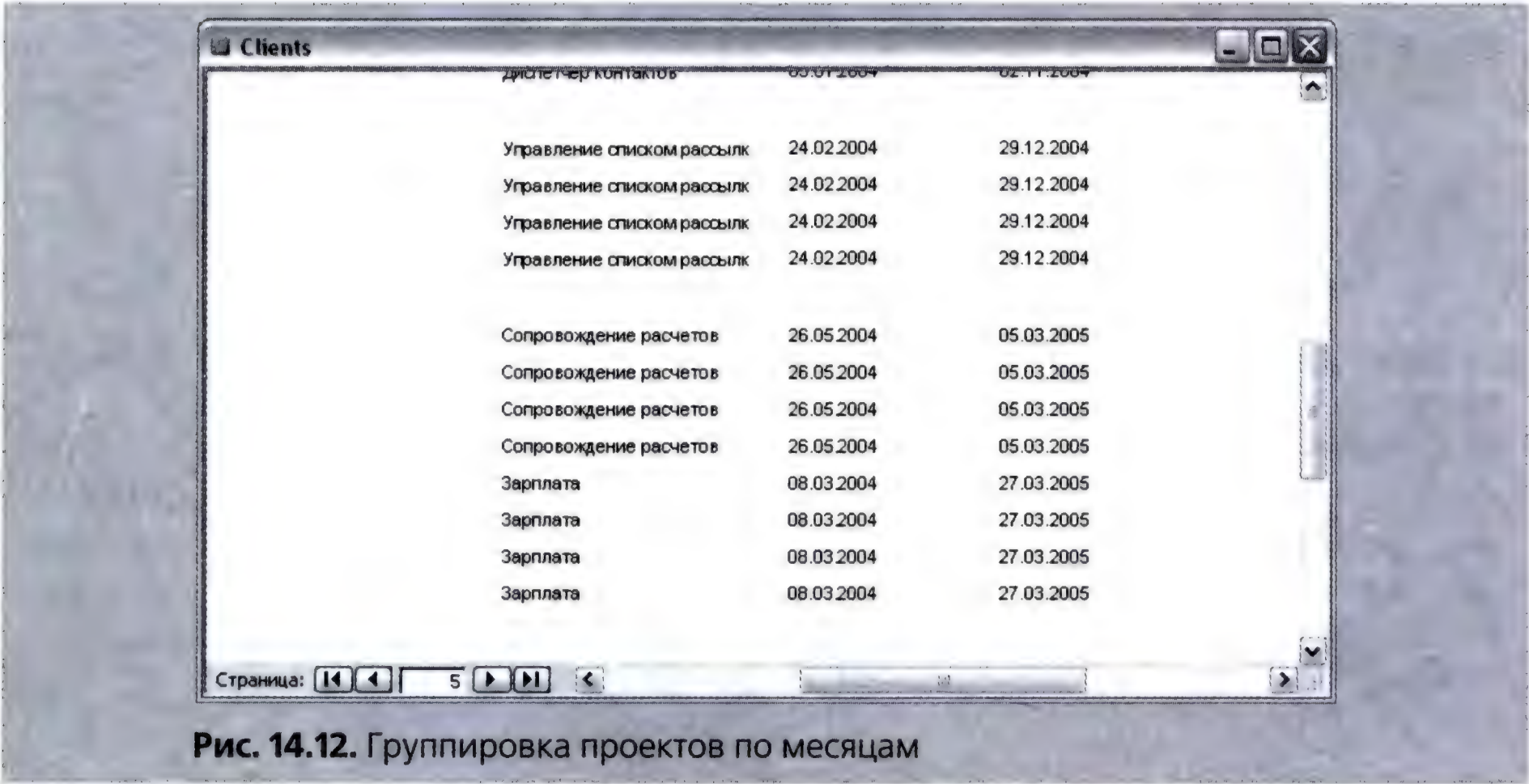


Рис. 14.12. Группировка проектов по месяцам

Работа с коллекциями

15

Коллекции Access

Вы уже знаете, что в любой базе данных Access содержатся всевозможные виды объектов: таблицы, формы, макросы и т.д. С некоторыми из этих объектов вы научились работать в коде VBA и теперь знаете, как открывать форму или просматривать отчет. Однако существуют и другие способы работы с объектами Access — с помощью встроенных коллекций приложения. На рис. 15.1 показан фрагмент объектной модели Access, содержащий эти коллекции.

Объект Application открывает два основных способа путешествия по объектной модели к отдельным объектам. Объект CurrentProject (Текущий проект) ведет к коллекциям всех объектов интерфейса пользователя: AllDataAccessPages (Все страницы доступа к данным), AllForms (Все формы) и т.д. Объект CurrentData (Текущие данные) ведет к коллекциям всех объектов, связанных с данными: AllQueries (Все запросы), AllTables (Все таблицы) и т.д. Каждый из этих контейнеров, в свою очередь, содержит объекты AccessObject (Объект Access), представляющие отдельные экземпляры в коллекции.

В ЭТОЙ ГЛАВЕ

Коллекции Access	263
Получение списка объектов.....	265
Работа со свойствами объектов	266
Программно определяемые зависимости	269

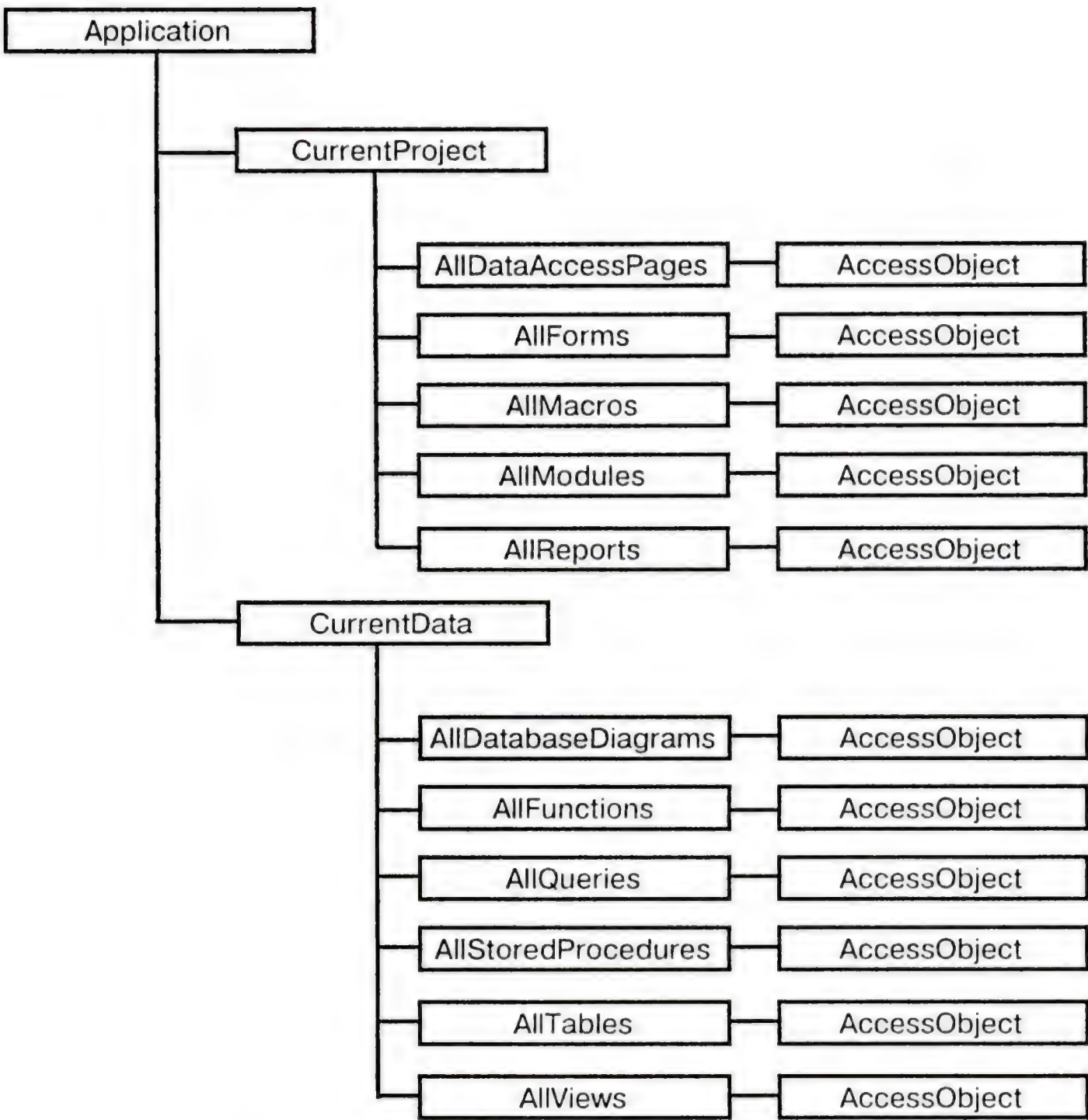


Рис. 15.1. Коллекции приложения и связанные объекты

Примечание

С технической точки зрения объект `CurrentProject` представляет объекты, созданные программой Access, в то время как объект `CurrentData` открывает доступ к объектам механизма базы данных. В обычных базах данных Access механизмом баз данных является *Microsoft Jet*, в то время как в проектах Access – *Microsoft SQL Server*. Большинство коллекций в объекте `CurrentData` (`AllDatabaseDiagrams`, `AllFunctions`, `AllStoredProcedures` и `AllViews`) применимы к проектам Access с помощью механизма *SQL Server*. Эти объекты мы не будем обсуждать в данной книге.

Предупреждение

Не путайте коллекции `AllForms` и `AllReports` с коллекциями `Forms` и `Reports`. В первой группе коллекций находятся все потенциально доступные соответствующие объекты всей базы данных, в то время как во второй – только открытые в настоящий момент.

В этой главе будет показано, что можно сделать с помощью этих объектов. Они открывают вашей программе путь к информации, которую вы видите в окне Базы данных Access при работе в интерфейсе пользователя. К тому же

они могут оказаться очень полезными при написании собственных инструментов в VBA. Каждая из коллекций объектов поддерживает четыре свойства:

- `Application` — указатель на родительский объект приложения `Application`.
- `Count` — количество объектов `AccessObject` в коллекции.
- `Item` — индексируемое свойство для обращения к отдельным объектам коллекции.
- `Parent` — указатель на родительский объект (например, на `CurrentData` или `CurrentProject`).

Примечание

В некоторых случаях свойство `Parent` может указывать на объекты `CodeProject` или `CodeData`. Эти объекты аналогичны объектам `CurrentProject` и `CurrentData`, однако ссылаются на базы данных, загруженные в программу. Вам не потребуется обращаться к этим базам данных при работе с обычными базами данных в интерфейсе пользователя; они представляют интерес только для разработчиков встраиваемых модулей.

Получение списка объектов

Одной из задач, для решения которых используют описанные выше коллекции, является задача получения списка объектов базы данных. На первый взгляд может показаться, что эту задачу не нужно программировать, поскольку все объекты и так видны в окне Базы данных Access. Однако вы вскоре увидите, что работа с собственными списками объектов позволит вам создавать для пользователей более дружелюбный интерфейс, чем тот, который предлагает незыблемое окно Базы данных.

К примеру, с помощью коллекции `AllForms` можно создать загрузчик форм общего назначения для базы данных примеров. Вот как это сделать:

1. Создайте новую форму в представлении конструктора. Назначьте форме заголовок `Список форм`. Поместите в форму элемент простого списка с именем `lstForms` и командную кнопку с именем `cmdOpen`. Установите свойство `Тип источника строк` элемента списка в `Список значений`.

2. Откройте модуль формы и введите следующий текст:

```
Private Sub cmdOpen_Click()
    ' Открытие выбранной формы
    If Not IsNull(lstForms.Value) Then
        DoCmd.OpenForm lstForms.Value
    End If
End Sub
```

```
Private Sub Form_Load()
```



```

' Наполнить список именами всех
' форм базы данных
Dim AO As AccessObject

For Each AO In CurrentProject.AllForms
    lbForms.AddItem (AO.Name)
Next AO

End Sub

```

3. Сохраните форму под именем FormList. Откройте форму FormList — в ней будут перечислены все формы базы данных. Выделите любую из форм в списке и щелкните на кнопке открытия формы (рис. 15.2).

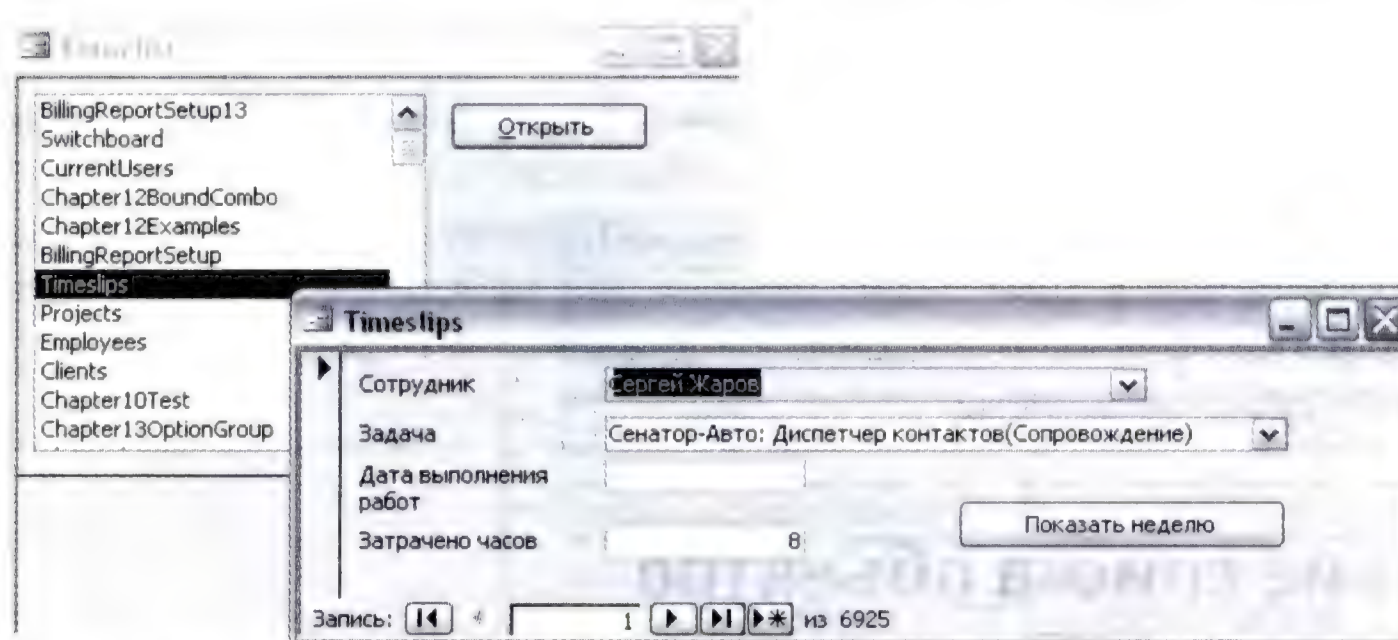


Рис. 15.2. В этой форме перечислены все формы базы данных

Работа со свойствами объектов

Как вы уже знаете, каждый из объектов во множестве коллекций представлен объектом класса `AccessObject`. Этот объект имеет ряд встроенных свойств, сообщающих о нем некоторые сведения:

- `CurrentView`. Константа, сообщающая текущее представление объекта (например, представления конструктора или таблицы), если объект открыт.
- `DateCreated`. Дата создания объекта.
- `DataModified`. Дата последнего изменения объекта.
- `FullName`. Полное имя (включая путь) объекта.
- `IsLoaded`. Если объект открыт в настоящий момент, то `True`, в противном случае — `False`.
- `Name`. Имя объекта.
- `Parent`. Коллекция, содержащая этот объект.
- `Properties`. Набор свойств объекта.
- `Type`. Константа, определяющая тип объекта (например, как форму, отчет или таблицу).

Примечание

Свойство `FullName` применяется только к страницам доступа к данным, которые хранятся как внешние файлы. Для всех остальных объектов оно пустое.

Объект `AccessObject` предлагает еще одну, дополнительную возможность: в этих объектах можно создавать дополнительные свойства. Это полезно практически для тех же целей, для которых создавалось свойство `Tag` в формах. Самостоятельно созданные свойства можно использовать для хранения дополнительной информации об объекте, используемой в программе.

➔ Более подробно свойство `Tag` описано в соответствующем разделе главы 13.

Рассмотрим форму `FormList` более пристально. В ее изначальной конструкции была заложена одна проблема: в ней перечислялись все формы, даже те, которые вы самостоятельно и не попытались бы открывать (к примеру, кнопочные панели, формы примеров и подформы). Все это кажется немного запутанным. Чтобы разобраться с этой проблемой, можно ввести дополнительное свойство, сообщаящее процедуре `FormList`, отображать ли данную форму в списке.

Для начала нам нужно добавить в формы новое свойство. Обычно такая работа осуществляется на этапе проектирования баз данных; пользователю нельзя разрешать корректировать это свойство. Поэтому для выполнения задачи следует добавить процедуру установки нового свойства в модуль примеров. Сама процедура может выглядеть следующим образом:

```
Public Sub ShowInFormList(strFormName As String)
    ' Пометка заданных форм, чтобы они
    ' отображались в форме FormList

    ' Извлекаем объект AccessObject,
    ' соответствующий заданной форме
    Dim AO As AccessObject
    Set AO = CurrentProject.AllForms(strFormName)

    ' Создаем в объекте новое свойство
    AO.Properties.Add "ShowInFormList", True

    Debug.Print "Свойство установлено"
End Sub
```

Каждый из объектов `AccessObject` имеет собственную коллекцию дополнительных свойств с именем `Properties`. В начале предложенной процедуры мы извлекаем объект `AccessObject`, соответствующий заданной форме. Затем с помощью метода `Add` коллекции `Properties` мы добавляем в объект новое свойство. Двумя аргументами этого метода являются имя нового свойства (представленное строкой) и его начальное значение (которое может иметь любой тип). В представленном примере процедуры создается новое свойство с именем `ShowInFormList`, начальным значением которого явля-

ется True. На рис. 15.3 показано, как вызывать данную процедуру в окне Immediate, чтобы добавить свойство ShowInFormList в форму Client.

Совет

Коллекция Properties объекта AccessObject содержит только дополнительные свойства, добавленные вами — она не содержит стандартных свойств, таких как Name или DateCreated.

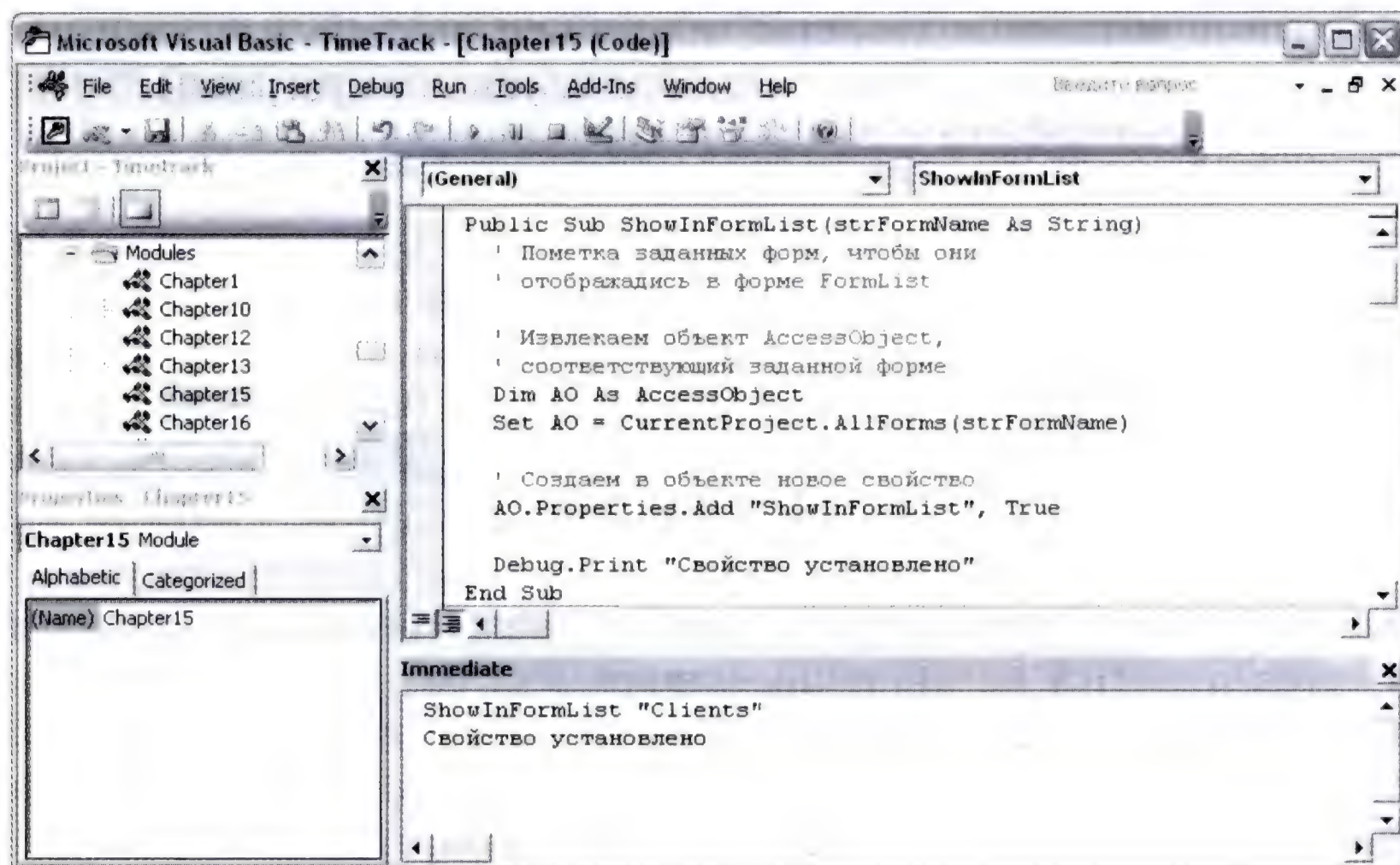


Рис. 15.3. Добавление в форму дополнительного свойства

В базе данных примеров мы использовали процедуру ShowInFormList для добавления дополнительного свойства в формы Clients, Employees, Projects и Timesleeps.

Следующим действием будет изменение процедур модуля формы FormList с целью отображения только тех форм, в которых дополнительное свойство установлено в True. Вот пересмотренный текст процедур:

```
Private Sub Form_Load()
    ' Наполнить список именами всех
    ' форм базы данных
    Dim AO As AccessObject

    ' Не прерываемся, если некоторое
    ' свойство не было найдено
    On Error Resume Next

    For Each AO In CurrentProject.AllForms
        If AO.Properties("ShowInFormList").Value = True Then
            If Err = 0 Then
                ' Добавляем только те формы, в которых данное
                ' свойство присутствует и равно True
                lbForms.AddItem (AO.Name)
            End If
        End If
    End For
End Sub
```



```

        End If
        Err.Clear
    End If
Next AO

```

```
End Sub
```

Работе с дополнительными свойствами следует уделять повышенное внимание. Если вы попытаетесь обратиться к несуществующему свойству, интерпретатор VBA вызовет ошибку. В приведенном фрагменте показан один из способов профилактики такой ситуации. Во-первых, мы вводим инструкцию `On Error Resume Next`, которая гарантирует, что никакая ошибка не станет фатальной. Во-вторых, мы пытаемся извлечь из всех форм значение дополнительного свойства. Здесь могут возникнуть три ситуации:

- Если свойство вообще не существует, выполнение будет передано следующему оператору, в то же время встроенная переменная `Err` будет установлена в некоторое значение, отличное от нуля. В этом случае строка добавления имени формы в список не выполняется.
- Если свойство существует и его значение равно `False`, не выполняется первое условие, и строка имени формы в список также не добавляется.
- Если же свойство существует и его значение равно `True`, проверка на ошибку вернет нуль, и имя формы будет добавлено в список.

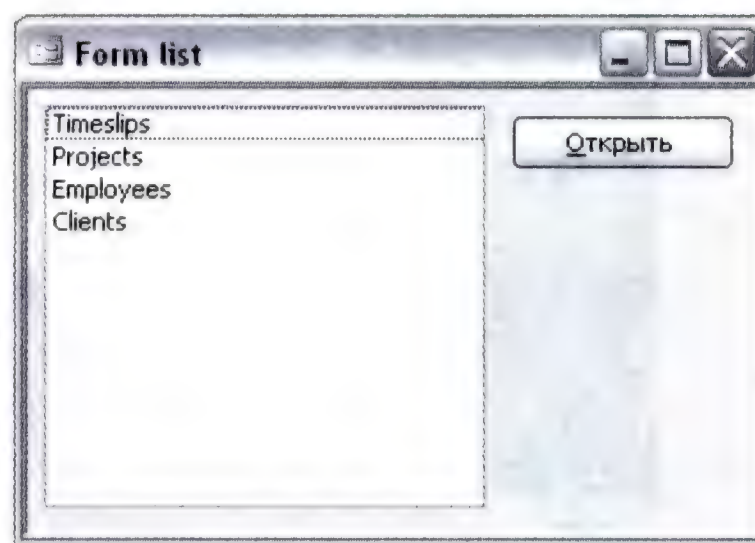


Рис. 15.4. Использование дополнительного свойства

На рис. 15.4 показана форма `FormList` после изменения текста ее процедур. Как вы видите, в ее списке перечислены только те формы, в которых дополнительное свойство существует и равно `True`.

Программно определяемые зависимости

Одни объекты Access могут зависеть от других. К примеру, форма может использовать запрос в качестве источника данных, а этот запрос, в свою очередь, может извлекать данные из одной или нескольких таблиц. В данном случае форма напрямую зависит от запроса и косвенно — от таблиц. Начиная с версии Access 2003, эта информация доступна через интерфейс пользователя, а также программно.

Если вы еще не обращались к этой информации в Access, вы ее легко найдете. Щелкните правой кнопкой на любом объекте в окне Базы данных и выберите в контекстном меню пункт **Зависимости объектов**. При этом откроется панель зависимостей, показанная на рис. 15.5. С помощью переключателя в

верхней части панели вы можете переключаться между отображением объектов, зависящих от данного, и объектов, от которых зависит этот объект.

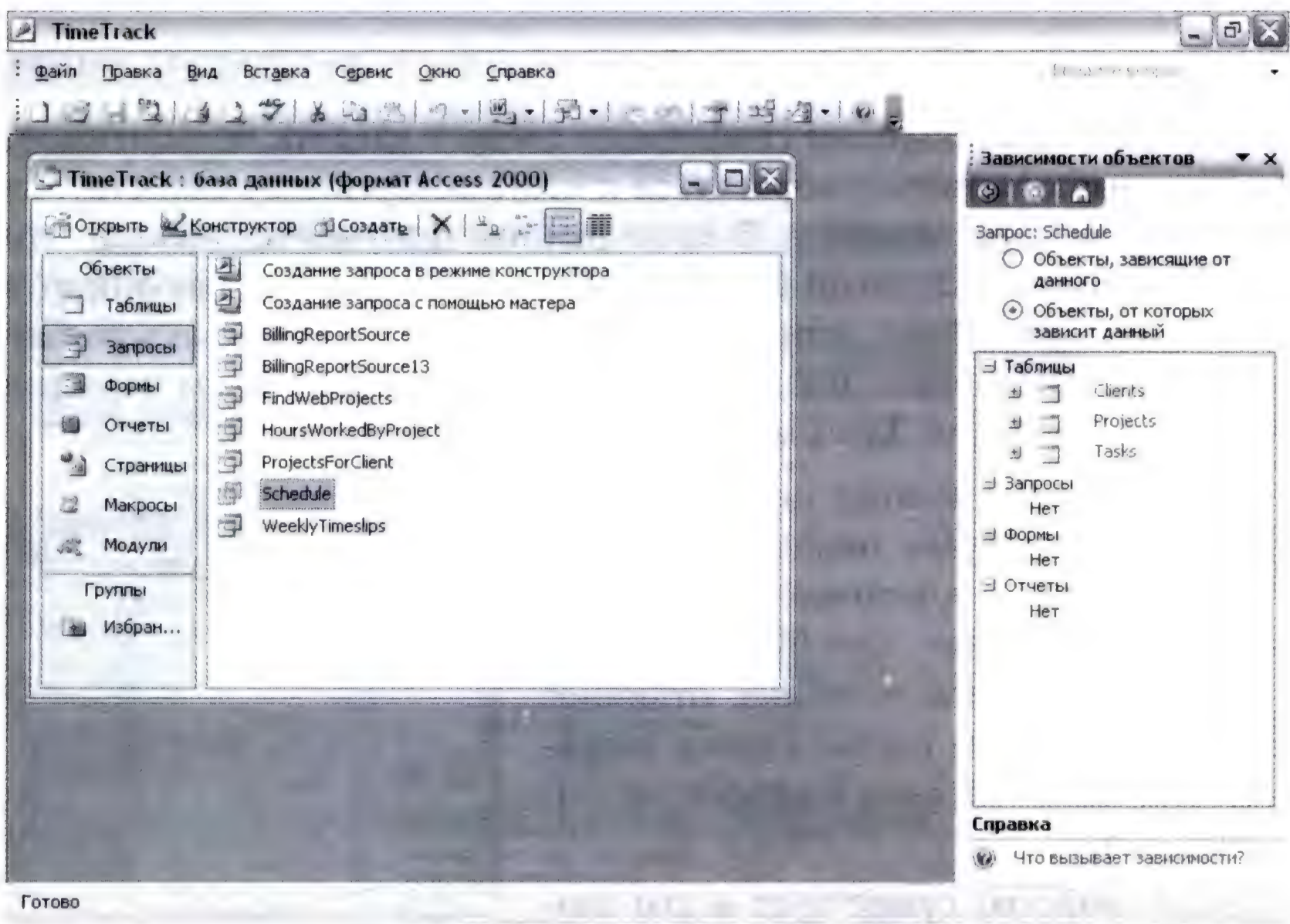


Рис. 15.5. Просмотр зависимостей объектов в интерфейсе пользователя

Предупреждение

Для того чтобы функция отображения зависимостей объектов работала, функция автозамены имен должна быть включена. Чтобы ее включить, нужно выполнить команду меню **Сервис**⇒**Параметры** (**Tools**⇒**Options**), после чего перейти к закладке **Общие** (**General**) и установить флажок **Отслеживать автозамену имен** (**Name AutoCorrect**). Следует отметить, что перед генерированием информации о зависимостях нужно сохранить и закрыть все открытые объекты.

Чтобы отследить зависимости объектов в VBA, необходимо пройти путь от объекта `AccessObject` до объекта `DependencyInfo`. Вот пример процедуры, которая выполняет эту работу:

```
Public Sub ShowDependencies(intType As AcObjectType, _
    strName As String)
    ' Показывает информацию о зависимостях заданного объекта

    Dim AO As AccessObject
    Dim AO2 As AccessObject
    Dim DI As DependencyInfo

    On Error GoTo HandleErr

    ' Извлекаем AccessObject
    Select Case intType
```



```

Case acTable
    Set AO = CurrentData.AllTables(strName)
    Debug.Print "Таблица: ";
Case acQuery
    Set AO = CurrentData.AllQueries(strName)
    Debug.Print "Запрос: ";
Case acForm
    Set AO = CurrentProject.AllForms(strName)
    Debug.Print "Форма: ";
Case acReport
    Set AO = CurrentProject.AllReports(strName)
    Debug.Print "Отчет: ";
End Select
Debug.Print strName

' Извлекаем информацию о зависимостях
Set DI = AO.GetDependencyInfo()

' Выводим результаты
If DI.Dependencies.Count = 0 Then
    Debug.Print "Этот объект не зависит ни от каких объектов"
Else
    Debug.Print "Этот объект зависит от следующих объектов:"
    For Each AO2 In DI.Dependencies
        Select Case AO2.Type
            Case acTable
                Debug.Print "Таблица: ";
            Case acQuery
                Debug.Print "Запрос: ";
            Case acForm
                Debug.Print "Формы: ";
            Case acReport
                Debug.Print "Отчет: ";
        End Select
        Debug.Print AO2.Name
    Next AO2
End If

If DI.Dependants.Count = 0 Then
    Debug.Print "От данного объекта не зависит ни один объект"
Else
    Debug.Print "От этого объекта зависят следующие:"
    For Each AO2 In DI.Dependants
        Select Case AO2.Type
            Case acTable
                Debug.Print "Таблица: ";
            Case acQuery
                Debug.Print "Запрос: ";
            Case acForm
                Debug.Print "Форма: ";
            Case acReport
                Debug.Print "Отчет: ";
        End Select
        Debug.Print AO2.Name
    Next AO2
End If

```



```
ExitHere:
    Exit Sub

HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description, vbCritical
    Resume ExitHere
End Sub
```

Этот фрагмент программы достаточно длинный, однако если вы пошагово его пройдете в режиме отладки, то сразу поймете, что он чрезвычайно прост. Первым делом мы извлекаем объект `AccessObject`, информация о зависимостях которого нас интересует. Процедура имеет два аргумента: тип объекта и его имя. Так как в Access существует перечисление `acObjectType`, содержащее всевозможные типы объектов, мы его можем использовать — это проще, чем создавать собственную классификацию. Первая конструкция `Select Case` с этой константой выполняет две вещи. Во-первых, в ней определяется, в какой коллекции искать корректный объект `AccessObject`. Во-вторых, в ней выводится в окно `Immediate` сообщение об имени и типе объекта.

Далее мы получаем объект `DependencyInfo`, который возвращается методом `GetDependencyInfo` объекта `AccessObject`. Объект `DependencyInfo` в своем составе имеет две коллекции, каждая из которых содержит объекты `AccessObject`. Коллекция `Dependencies` содержит объекты, от которых зависит данный; коллекция `Dependants` — объекты, зависящие от данного.

В оставшейся части программы мы проходим по этим двум контейнерам и выводим в окне `Immediate` их содержимое. Вот результат выполнения этой процедуры на объекте запроса `Schedule` базы данных примеров `TimeTrack`:

```
ShowDependencies acQuery, "Schedule"
Запрос: Schedule
Этот объект зависим от следующих объектов:
    Таблица: Clients
    Таблица: Projects
    Таблица: Tasks
От этого объекта зависят следующие:
    Отчет: Schedule
```

ПРАКТИКУМ

Расширение главной формы

В качестве демонстрации того, как контейнеры объектов можно использовать в интерфейсе пользователя, попробуем расширить форму `MasterForm` (см. главу 13) и включить в нее отчеты. Мы добавим в нее элемент списка всех отчетов и кнопку, вызывающую печать выбранного. Здесь казус заключается в том, что некоторые отчеты нельзя вывести, не введя дополнительную информацию. В базе данных `TimeTrack` отчет `BillingReport` требует предварительного запуска формы `BillingReportSetup` для введения нужных данных. Это требование можно учесть, введя в отчет дополнительное свойство.

В нашей базе данных существует всего один отчет, требующий дополнительного свойства. Для его добавления нужно будет написать процедуру — эту операцию можно выполнить непосредственно в окне **Immediate**. В приведенном ниже примере мы добавляем дополнительное свойство `NeedForms` и присваиваем ему имя необходимой формы — `BillingReportSetup`:

```
CurrentProject.AllReports("BillingReport")._
    Properties.Add "NeedsForm", "BillingReportSetup"
```

Следующим действием будет добавление нужных элементов на форму `MasterForm`. Назовем элемент списка именем `lstReports`, а командную кнопку именем `cmdOpenReport`. При этом нужно будет немного увеличить размеры формы по высоте и добавить соответствующие элементы. Теперь нужно установить свойство **Тип источника строк списка** в **Список значений**.

Осталось только добавить в модуль формы процедуры наполнения списка и реакции на нажатие командной кнопки:

```
Private Sub Form_Load()
    ' Наполнение списка именами
    ' всех отчетов в базе данных
    Dim AO As AccessObject

    For Each AO In CurrentProject.AllReports
        lstReports.AddItem (AO.Name)
    Next AO

End Sub

Private Sub cmdOpenReport_Click()
    ' Открытие выбранной формы или отчета
    Dim AO As AccessObject
    Dim strForm As String

    On Error Resume Next

    If Not IsNull(lstReports.Value) Then
        ' Вызов соответствующего объекта AccessObject
        Set AO = CurrentProject.AllReports(lstReports.Value)
        ' Проверка на наличие дополнительного свойства
        strForm = AO.Properties("NeedsForm")
        If Err = 0 Then
            ' Свойство присутствует. Открываем форму
            DoCmd.OpenForm strForm
        Else
            ' Свойство отсутствует. Открываем отчет
            DoCmd.OpenReport AO.Name, acViewPreview
        End If
    End If

End Sub
```

При загрузке формы `MasterForm` данная процедура проходит по всем элементам контейнера `AllReports` и наполняет именами отчетов список формы. Если у вас есть какие-либо основания скрывать из списка отдельные отчеты, вы можете воспользоваться той же методикой, которую мы применяли в главе 13 к элементу списка форм.

Когда пользователь щелкает на кнопке открытия отчета, процедура обработки щелчка вызывает объект `AccessObject`, представляющий данную позицию списка. После этого производится попытка считать свойство `NeedsForm`, и если некоторое значение возвращается, открывается форма с этим именем; в противном случае в режиме предварительного просмотра открывается отчет.

На рис. 15.6 показана завершенная форма `MasterForm`. Если открыть отчет `Schedule`, он напрямую отобразится в окне предварительного просмотра; если же попытаться открыть отчет `BillingReport`, вначале откроется форма запроса параметров отчета.

The screenshot shows a window titled "MasterForm : форма". At the top, there are three tabs: "Клиенты" (Clients), "Сотрудники" (Employees), and "Проекты" (Projects). The "Клиенты" tab is active, displaying a form with the following fields:

- Клиент: Сенатор-Авто
- Адрес: Ленинградский пр. 37, корп.2
- Город: Москва
- Регион: Россия
- Индекс: 125315
- Тел.: 095-155-6610
- Контакт: Михаил Кузнецов

Below the form fields, there is a record navigation section labeled "Запись:" with buttons for navigating between records (back, forward, first, last, search) and a text box showing "1 из 11".

At the bottom, there is a list of reports: "BillingReport14", "Schedule", "BillingReport", and "BillingReport13". To the right of this list is a button labeled "Отчет" (Report).

Рис. 15.6. Усовершенствованная форма MasterForm



средствами ADO

16

Что такое библиотека
ADO и зачем она нужна

В ЭТОЙ ЧАСТИ

Работа с данными в Access

- 16. Извлечение данных средствами ADO
- 17. Манипуляции с данными средствами ADO
- 18. Создание объектов средствами ADOX
- 19. Расширенные операции с данными

Извлечение данных средствами ADO

16

Что такое библиотека ADO и зачем она нужна

Данные, вводимые в базу данных Access, на самом деле не хранятся в объектах базы данных. Формы, таблицы, отчеты и даже запросы являются всего лишь интерфейсными объектами, с помощью которых осуществляется ввод, просмотр и извлечение данных. За кулисами приложение Access занимается вопросами обмена данными — вам совершенно не требуется знать механизмы этих операций и даже факт самого их существования. И в этом состоит очевидное достоинство приложения Access.

Все становится немного сложнее, когда для взаимодействия с данными вы начинаете использовать программы. Именно здесь выходит на сцену библиотека объектов данных *ADO* (ActiveX Data Objects) — объектная модель, содержащая ряд специализированных объектов извлечения данных. Библиотека ADO — это ваш пропуск ко всем типам данных, и не только в Access.

Устанавливать библиотеку ADO нет необходимости — начиная с версии Windows 2000 эта библиотека устанавливается как часть операционной системы. К тому же библиотека ADO использовалась по умолчанию в Access начиная с версии Access 2000, так что для использования ее объектов вам даже не придется устанавливать ссылки на нее в приложении.

В ЭТОЙ ГЛАВЕ

Что такое библиотека ADO и зачем она нужна	277
Использование объекта ADO Connection	278
Работа с объектами Command	283
Различные типы наборов записей	285
Создание и открытие набора записей	286
Фильтрация набора записей ...	287
Использование свойства Recordset	289

Объектная модель ADO

Подобно другим объектным моделям, библиотека ADO — это иерархическая система объектов. Рисунок 16.1 наглядно иллюстрирует объектную модель ADO. Эта модель состоит из коллекций специфических типов объектов. Ядром объектной модели ADO является объект `Connection` (Подключение), который представляет одно подключение к источнику данных *OLE DB*. В этой главе мы рассмотрим следующие объекты модели ADO: `Connection`, `Command` (Команда) и `Recordset` (Набор данных).

Примечание

Библиотеки ADO и OLE DB работают совместно. OLE DB связывается с данными и переводит компоненты обработки данных из одного формата в другой с помощью двух типов компонентов: *поставщиков* и *потребителей*. Поставщики являются программами, которые общаются друг с другом, представляя данные. Потребители используют представленные данные. В этой схеме ADO выступает потребителем. Другими словами, подключение к OLE DB позволяет подключиться к самим данным, а объекты ADO позволяют извлекать данные после подключения.

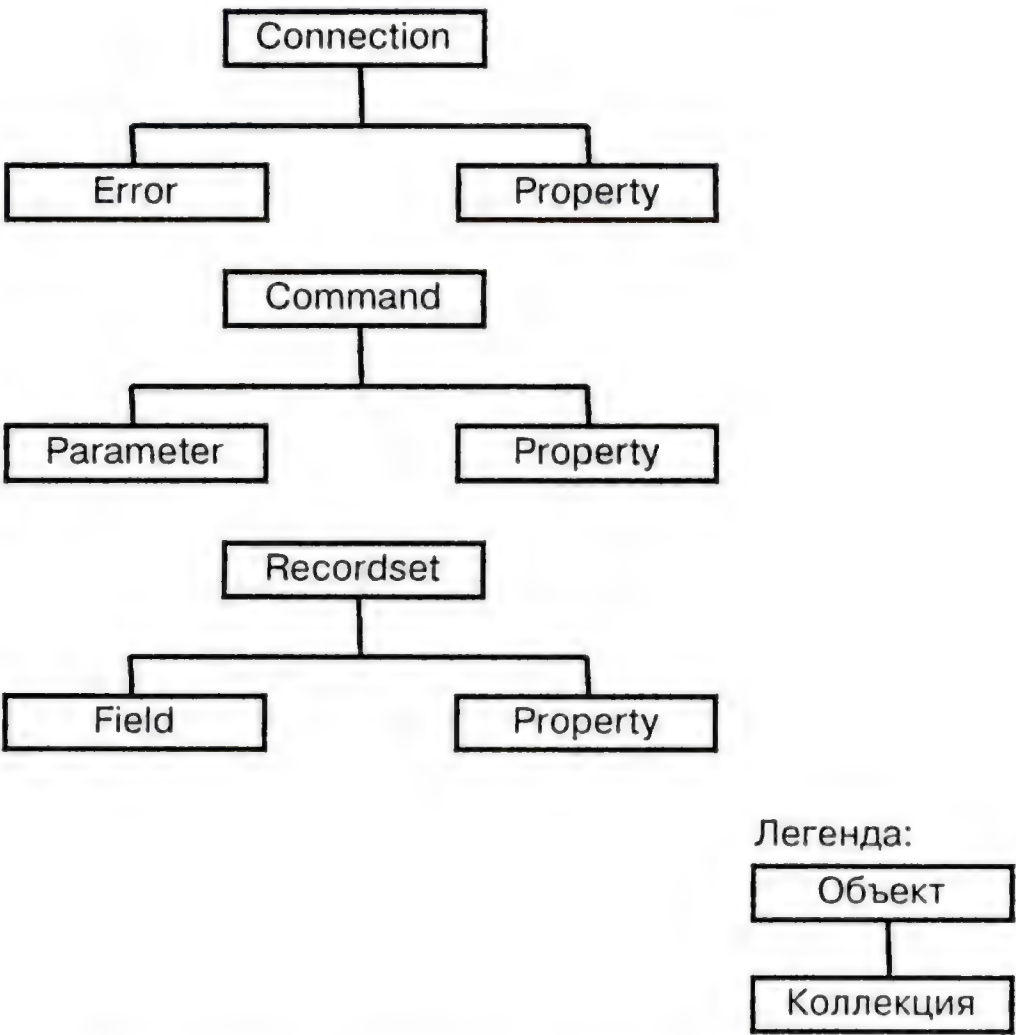


Рис. 16.1. Объектная модель ADO состоит из объектов

Использование объекта ADO Connection

Первое, что предстоит сделать, — это подключиться к источнику данных и объекту, в котором хранятся необходимые данные. С технической стороны, объект `Connection` представляет собой одно подключение к источнику дан-

ных OLE DB. Это значит, что подключение лучше всего осуществлять с его помощью.

Подключаться можно как явно, так и неявно, и ни один из этих методов перед другим не имеет преимуществ. Однако если одно и то же подключение вы собираетесь использовать неоднократно, используйте *явный* объект `Connection`. *Неявное* подключение создается, когда вы запрашиваете данные, предварительно не создав объект `Connection`. Для большей ясности в первое время всегда используйте явный объект `Connection`, пока не начнете свободно ориентироваться в объектной модели ADO.

Открытие подключения

При явном подключении объект `Connection` объявляется, после чего создается его экземпляр:

```
Dim cnn As ADODB.Connection
Set cnn = New ADODB.Connection
```

Неявное подключение невозможно показать по той причине, что его объявления просто не существует.

Объект `Connection` имеет целый ряд свойств, управляющих его поведением:

- `ConnectionString`. В этой строке задается источник данных.
- `ConnectionTimeout`. Здесь определяется время ожидания (в секундах) ответа источника данных. По умолчанию принято 15 секунд, однако это время вы можете продлить, особенно при работе в условиях загруженной сети или при передаче данных через Internet.
- `Mode`. Это свойство определяет режим доступа. В табл. 16.1 перечислены внутренние константы, используемые для этого свойства.
- `CursorLocation`. Определяет положение курсора OLE DB. Принимает значения двух констант: `adUseServer` устанавливает курсор на стороне сервера, `adUseClient` — на стороне клиента. Курсор вы можете себе представить как множество строк из таблицы плюс индикатор текущей строки.
- `DefaultDatabase`. Определяет базу данных на сервере, которая будет использоваться для всех процессов данных в объекте `Connection`.
- `IsolationLevel`. Это свойство управляет тем, как операции с данными в разных подключениях влияют друг на друга. В табл. 16.2 перечислены встроенные константы, используемые для данного свойства. Этот параметр вы будете применять тогда, когда начнете писать программы, используемые несколькими пользователями для одновременного обращения к одной и той же базе данных.
- `Provider`. Определяет поставщика OLE DB перед открытием подключения.

- `CommandTimeout`. Определяет длительность ожидания при попытке подключения, связанным с выполнением некоторой команды; после этого попытка завершается.

Таблица 16.1. Константы свойства `Mode`

Константа	Описание
<code>acModeRead</code>	Подключение только для чтения
<code>adModeReadWrite</code>	Подключение для чтения и записи
<code>adModeWrite</code>	Подключение только для записи
<code>adModeShareDenyRead</code>	Другие приложения не могут подключаться для чтения
<code>adModeShareDenyWrite</code>	Другие приложения не могут подключаться для записи
<code>adModeShareDenyNone</code>	Все приложения могут подключаться с любыми правами доступа
<code>adModeShareExclusive</code>	Другое приложение не может подключаться ни с какими правами
<code>adModeRecursive</code>	Подзаписи наследуют разрешения текущей записи

Таблица 16.2. Константы свойства `IsolationLevel`

Константа	Описание
<code>adXactUnspecified</code>	Возвращается, когда поставщик не может определить уровень изоляции
<code>adXactChaos</code> <code>adXactBrowse</code>	Установки по умолчанию, защищающие незаконченные изменения от перезаписи
<code>adXactReadUncommitted</code> <code>adXactCursorStability</code>	Позволяют просматривать, но не редактировать неподтвержденные изменения от других транзакций
<code>adXactReadCommitted</code>	Позволяет просматривать изменения от других транзакций только после их подтверждения
<code>adXactRepeatableRead</code> <code>adXactIsolated</code>	Позволяет обновлять набор данных при внесении изменений другими транзакциями
<code>adXactSerializable</code>	Изолирует все транзакции от всех других транзакций

Перед открытием подключения не забудьте установить все его свойства. К примеру, чтобы установить курсор на стороне сервера, можно использовать следующий фрагмент программы:

```
Dim cnn As ADODB.Connection
Set cnn = New ADODB.Connection
cnn.CursorLocation = adUseServer
```


Для открытия подключения используется метод `Open` объекта `Connection`, имеющий следующий синтаксис:

```
Connection.Open [строка_подключения] [, идентификатор_пользователя] _  
[, пароль] [, параметры]
```

Все аргументы этого метода являются не обязательными. Аргумент *параметры* может принимать значение двух встроенных констант: `adConnectUnspecified` — для открытия синхронного соединения (одновременно может происходить только одна операция) (принято по умолчанию) или `adAsyncConnect` — для открытия асинхронного соединения. Использование синхронного соединения, принятого по умолчанию, отлично подойдет для всех операций, которые не требуют больших объемов данных.

Строка подключения

Для определения строки подключения существует две возможности:

- использование аргумента *строка_подключения* метода `Open` после создания объекта `Connection`;
- использование свойства `ConnectionString` объекта `Connection` после его создания, но до установки физического подключения.

Как бы вы ни поступили, в виде пяти аргументов должна быть передана определенная информация, которая впоследствии будет объединена в строку подключения.

- **Поставщик.** Определяет имя поставщика OLE DB (точная строка указана в документации к данному поставщику). В табл. 16.3 представлен список строк, соответствующих наиболее распространенным поставщикам.
- **Источник данных.** Определяется файл, к которому производится подключение.
- **Удаленный поставщик.** Определяется поставщик сервера, если открывается подключение на стороне клиента.
- **Удаленный сервер.** Идентификация сервера.
- **URL.** Строка подключения определяется как адрес URL.

Таблица 16.3. Строки наиболее популярных поставщиков

Приложение	Строка
Microsoft Jet 3.51	Microsoft.Jet.OLEDB.3.5.1
Microsoft Jet 4.0	Microsoft.Jet.OLEDB.4.0
Драйвер ODBC	MSDASQL
Oracle	MSDAORA
SQL Server	SQLOLEDB

Совет

Если вам нужна помощь в создании строки подключения, зайдите на узел <http://www.connection-string.com/>, и вы получите ответы на все свои вопросы.

Приведенная в следующем примере строка производит подключение к базе данных примеров Northwind, поставляемой в комплекте установки Access (предполагается, что место установки использовано по умолчанию).

```
Private Sub MakeConnection()
    Dim cnn As ADODB.Connection
    Dim strConn As String
    Set cnn = New ADODB.Connection
    strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\Program Files\Microsoft Office\" & _
        "Office\Samples\Northwind.mdb;"
    cnn.Open strConn
    MsgBox "Соединение установлено"
    cnn.Close
    Set cnn = Nothing
End Sub
```

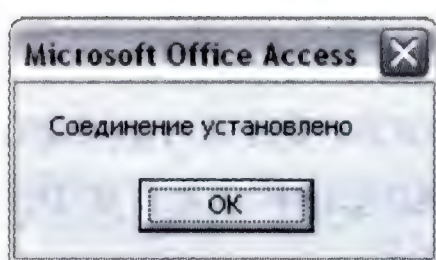


Рис. 16.2. В этом окне сообщается, что соединение с базой данных установлено

Откройте стандартный модуль и введите эту процедуру, после чего нажмите клавишу <F5>. Когда подключение к базе данных будет установлено, откроется окно сообщения, показанное на рис. 16.2. Щелкните на кнопке ОК, и окно закроется. В приведенном примере источник данных устанавливается в открываемую базу данных. (В аргументе `Data Source` вы можете задать любую базу данных; просто старайтесь описать точный и полный путь.)

Заккрытие подключения

В приведенном выше примере мы не выполняли никаких действий, кроме открытия источника данных. Никакие данные не извлекались и не обрабатывались. В то же время вы, наверное, обратили внимание на наличие инструкции `Close` в конце процедуры. Метод `Close` объекта `Connection` позволяет разорвать соединение с источником данных. Сам объект можно не уничтожать, а использовать впоследствии для других целей.

Метод `Close` имеет следующий синтаксис:

```
соединение.Close
```

где *соединение* — это имя объекта `Connection`.

Для повторного использования объекта `Connection` нужно еще раз выполнить его метод `Open`. Когда работа с объектом `Connection` завершена, после закрытия соединения ему нужно присвоить значение `Nothing`:

```
Set соединение = Nothing
```


Совет

Чаще всего строку соединения можно упростить. Если необходимые данные находятся в текущей базе данных, можно использовать объект `CurrentProject` и подключиться к ней следующим образом:

```
Dim cnn As New ADODB.Connection
Set cnn = CurrentProject.Connection
```

Используя этот метод, вы подключаетесь к данным, открытым в настоящий момент в Access.

Работа с объектами `Command`

Для извлечения данных и манипуляций с ними можно использовать объекты `Connection` и `Command`. Однако последний объект имеет ряд преимуществ:

- Используя объект `Command`, можно избежать создания набора данных (объект `Recordset`) и обновлять данные напрямую. При работе с некоторыми базами данных это может значительно ускорить процесс, так как все изменения будут совершаться непосредственно на сервере, и записи не будут перемещаться по сети.
- Объект `Command` поддерживает параметры, в отличие от объекта `Connection`.
- Объект `Command` имеет больше свойств, чем `Connection`, что позволяет более точно настраивать его поведение.

Объект `Command` в чем-то можно сравнить с хранимыми процедурами и другими объектами доступа к данным, так как он реализует процесс, реально возвращающий данные. Это производится путем выполнения программных инструкций над источником данных OLE DB или путем извлечения этих данных. Все сказанное делает объект `Command` достаточно гибким инструментом в руках программиста.

Создание объекта `Command`

Создание объекта `Command` ничем не отличается от создания объекта `Connection` — его нужно объявить и затем определить. В то же время данный объект не нужно открывать и в этом состоит существенное удобство его использования. Для объявления и определения объекта `Command` используют следующую структуру:

```
Dim strConn As String
Dim cmd As ADODB.Command
strConn = строка_подключения
Set cmd = New ADODB.Command
cmd.ActiveConnection = strConn
```

Аргумент *строка_подключения* идентичен тому, который используется в объекте `Connection`.

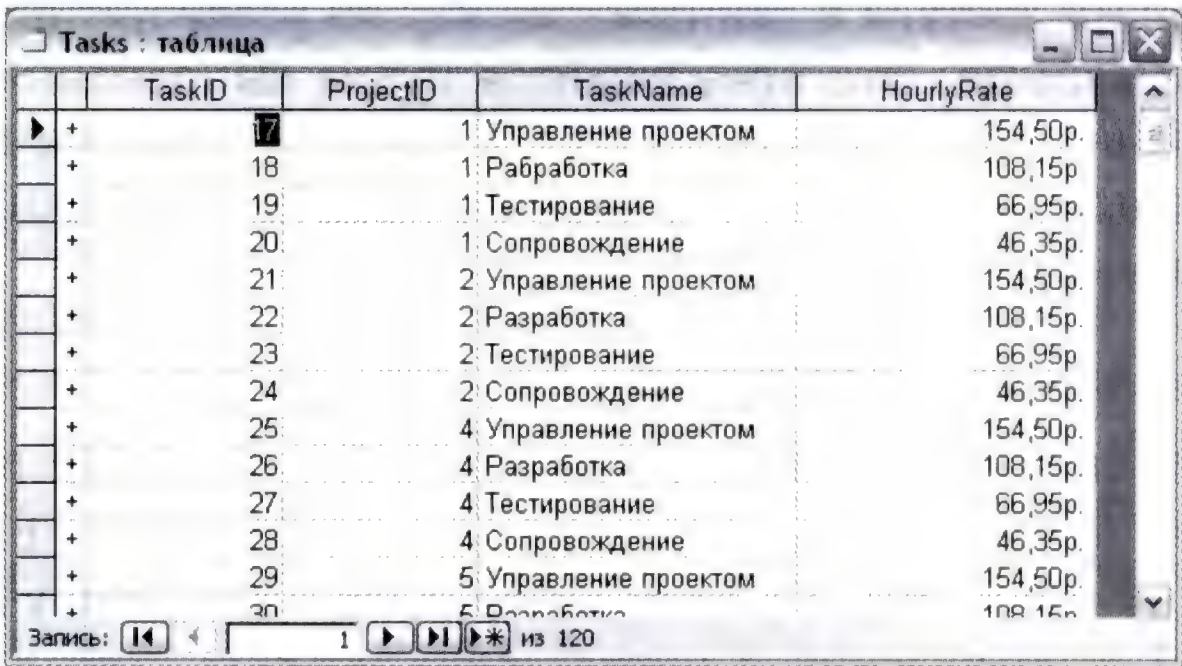
Выполнение команд

Объект Command позволяет выполнять команды манипуляций данными в источнике данных — можно как изменять данные, так и просто извлекать их. Как правило, для извлечения данных используют объект Recordset, который рассматривается далее.

Для выполнения программ, обновляющих или извлекающих данные, используется свойство CommandText объекта Command. В нем содержатся фактические инструкции, выполняемые в источнике данных (говоря техническими терминами, в поставщике). Это свойство является строковым выражением или значением, содержащим команду поставщика (это может быть выражение SQL, имя таблицы, URL или вызов хранимой процедуры).

Приведенная ниже процедура автоматизирует простой запрос обновления, увеличивающий на три процента часовую ставку в таблице Tasks базы данных TimeTrack; при этом используется выражение UPDATE языка SQL.

```
Private Sub EditCA()  
    'Увеличение часовой ставки на три процента  
    Dim strConn As String  
    Dim cmd As ADODB.Command  
    Set cmd = New ADODB.Command  
    With cmd  
        .ActiveConnection = CurrentProject.Connection  
        .CommandText = "UPDATE Tasks " &  
            "SET HourlyRate = HourlyRate + (HourlyRate * .03) "  
        .Execute  
    End With  
    Set cmd = Nothing  
End Sub
```



The screenshot shows the 'Tasks' table in Microsoft Access. The table has four columns: TaskID, ProjectID, TaskName, and HourlyRate. The data is as follows:

TaskID	ProjectID	TaskName	HourlyRate
17	1	Управление проектом	154,50р.
18	1	Работа	108,15р.
19	1	Тестирование	66,95р.
20	1	Сопровождение	46,35р.
21	2	Управление проектом	154,50р.
22	2	Разработка	108,15р.
23	2	Тестирование	66,95р.
24	2	Сопровождение	46,35р.
25	4	Управление проектом	154,50р.
26	4	Разработка	108,15р.
27	4	Тестирование	66,95р.
28	4	Сопровождение	46,35р.
29	5	Управление проектом	154,50р.
30	5	Разработка	108,15р.

At the bottom of the window, the status bar indicates 'Запись: 1 из 120' (Record: 1 of 120).

Рис. 16.3. Объект Command обновил часовые ставки

Введите эту процедуру в стандартный модуль (или используйте готовую процедуру в модуле Chapter16 базы данных TimeTrack). Установите курсор внутри тела процедуры и нажмите клавишу <F5>. На экране ничего не произойдет. А теперь откройте таблицу Tasks и вы увидите увеличенные значения часовых ставок работ. К примеру, первая работа Управление проектом, ставка которой соответствовала “150р.”, теперь имеет ставку “154,50р.”.

Различные типы наборов записей

Непосредственное изменение данных в источнике — быстрый и эффективный путь, если нет необходимости постоянного отслеживания и проверки данных. Иногда данные нужно извлечь, например, для того, чтобы проанализировать оклады сотрудников и принять решение относительно их изменения.

Обычно данные извлекаются с помощью объекта `Recordset` (Набор данных). Этот объект можно представить как некоторый контейнер данных, позволяющий выполнять команды объектов `Connection` и `Command`. Набор данных имеет практически тот же вид, что и таблица базы данных Access, открытая в табличном представлении.

С технической стороны, объект `Recordset` представляет собой *курсор*. Курсор часто описывают в литературе как указатель на текущую запись, но на самом деле он представляет собой нечто большее. Это — все извлеченные данные с указателем, установленным на текущую запись. Существует четыре типа курсоров (объектов `Recordset`):

- *Динамический курсор* позволяет видеть в режиме реального времени изменения, выполненные другими пользователями; к тому же сам указатель можно перемещать в любом направлении. Механизм Jet не поддерживает этот вид курсора, и вместо динамического устанавливает ключевой курсор.
- *Ключевой курсор* аналогичен динамическому, однако не отражает добавление и удаление записей другими пользователями. Изменения текущих данных в этом курсоре отражаются. Указатель можно перемещать в любом направлении. В однопользовательском режиме динамический и ключевой курсоры идентичны.
- *Статический курсор* создает копию данных. Вы не можете изменить данные или увидеть изменения, выполняемые другими пользователями. Обеспечивает перемещение по записям во всех направлениях.
- *Курсор последовательного доступа* является статическим курсором с ограниченными перемещениями: по записям можно перемещаться только вперед. Этот курсор устанавливается в ADO по умолчанию.

Выбор типа курсора обычно зависит от данных. Динамический курсор более гибок, однако требует больших ресурсов (к тому же он не доступен в Access). Динамические курсоры обычно замедляют выполнение операций. В то же время курсор последовательного доступа реагирует быстрее, однако является самым ограниченным из всех четырех типов — данные невозможно изменить, к тому же по записям можно перемещаться только вперед.

Предупреждение

Механизм Microsoft Jet не поддерживает динамический курсор, хотя в библиотеке ADO вы можете его запросить. Если вы это сделаете, механизм Jet вернет ключевой курсор, который не будет отражать в реальном времени записи, добавляемые другими пользователями, как это происходит в реальном динамическом курсоре. Если вы отслеживаете изменения данных, просто чаще обновляйте курсор.

Создание и открытие набора записей

Объект набора записей `Recordset` создается с помощью ключевого слова `Dim`:

```
Dim набор_записей As ADODB.Recordset
Set набор_записей = New ADODB.Recordset
```

Наполнение набора записей ничем не сложнее его открытия. Для этого используется метод `Open`, имеющий следующий синтаксис:

```
набор_записей.Open источник [, активное_соединение] [, тип_курсора] _
    [, тип_блокировки] [, параметры]
```

где `набор_записей` — имя объекта `Recordset`. В табл. 16.4 перечислены остальные аргументы этого метода.

Таблица 16.4. Аргументы объекта `Recordset`

Аргумент	Описание
<i>источник</i>	Идентифицирует данные, которые используются для наполнения объекта <code>Recordset</code> . Этим аргументом может быть имя объекта <code>Command</code> , выражение SQL, имя таблицы или хранимой процедуры, уже существующий набор записей, объект потока <code>Stream</code> или URL
<i>активное_соединение</i>	Идентифицирует соединение, используемое для захвата данных, определенных <i>источником</i> . Его значением может выступать объект <code>Connection</code> или строка подключения
<i>тип_курсора</i>	Определяет тип курсора. Может принимать значение одной из четырех констант: <code>adOpenDynamic</code> , <code>adOpenKeyset</code> , <code>adOpenStatic</code> или <code>adOpenForwardOnly</code>
<i>тип_блокировки</i>	Определяет тип блокировки и принимает значение одной из констант, перечисленных в табл. 16.5
<i>параметры</i>	Дополнительная информация, которую может требовать поставщик. В табл. 16.6 перечислены константы, допустимые для передачи в этот аргумент.

Таблица 16.5. Константы аргумента *тип_блокировки*

Константа	Описание
<code>adLockReadOnly</code>	Создание доступного только для чтения набора записей (используется по умолчанию)
<code>adLockPessimistic</code>	Записи блокируются на время редактирования
<code>adLockOptimistic</code>	Записи блокируются на время обновления
<code>adLockBatchOptimistic</code>	Используется совместно с методом <code>UpdateBatch</code> для обновления нескольких записей за одну операцию

Таблица 16.6. Константы аргумента параметры

Константа	Описание
adCmdUnknown	Отсутствие дополнительной информации (используется по умолчанию)
adCmdText	Определяет свойство CommandText как хранимую процедуру (набор команд, а не имя процедуры)
adCmdTable	Определяет свойство CommandText как имя таблицы
adCmdStoredProc	Определяет свойство CommandText как имя хранимой процедуры
adCmdFile	Определяет свойство CommandText как имя файла
adCmdTableDirect	Определяет свойство CommandText как имя таблицы. Возвращаемый набор записей является единственным типом, поддерживающим метод поиска Seek
adAsyncExecute	Асинхронное выполнение команды
adAsyncFetch	Определяет синхронный способ наполнения кэша, после чего дополнительные строки выбираются асинхронно
adAsyncFetchNonBlocking	Асинхронное извлечение записей (если это возможно) без блокирования основного потока

Как видите, перед открытием объекта Recordset есть о чем подумать. Однако хорошо то, что все операции достаточно просты. К примеру, в следующем фрагменте программы объект набора записей наполняется содержимым таблицы Clients.

```
Private Sub ClientsRst()  
    Dim rst As ADODB.Recordset  
    Set rst = New ADODB.Recordset  
    rst.Open "Clients", CurrentProject.Connection  
    MsgBox "Соединение установлено"  
    rst.Close  
    Set rst = Nothing  
End Sub
```

Если вы решили запустить эту процедуру из стандартного модуля, то окажется, что она на самом деле ничего не делает. Да, в ней создается объект набора записей rst, но с данными, содержащимися в этом наборе, не выполняются никаких операций. В то же время данный метод является самым сжатым методом создания объекта набора записей.

Фильтрация набора записей

Поиск конкретных данных в объекте Recordset легче всего выполнять с помощью свойства Filter. Вы можете выполнить новый запрос или даже

использовать метод `Find` (о нем мы поговорим в следующей главе), однако свойство `Filter` ближе к задаче поиска по критерию. Выполнение нового запроса полностью заменяет набор записей, а метод `Find` позволяет находить записи только по одной. Свойство/метод `Filter` временно ограничивает доступ только к тем записям, которые удовлетворяют введенному критерию. Другими словами, вы получаете подмножество записей, содержащихся в объекте `Recordset`.

Совет

Если вы только начинаете изучение библиотеки ADO, будьте внимательны при создании объектов наборов записей. По умолчанию в ADO создаются наборы записей последовательного доступа, что отличает ее от DAO. В DAO по умолчанию создается таблица, далее следуют динамический набор данных, слепок данных и только в заключение — набор последовательного доступа. В ADO наборы записей с последовательным доступом остались только из соображений совместимости. Одна из проблем, которая может возникнуть при использовании умолчаний, связана с подсчетом записей. Набор записей с последовательным доступом не имеет свойства количества записей (`RecordCount`). Если вам нужно это значение для работы, открывайте ключевой или статический курсор.

Для того чтобы временно ограничить число доступных в наборе данных записей, используйте свойство `Filter` в следующем виде:

```
набор_записей.Filter = условие
```

где *набор_записей* — имя объекта `Recordset`, а *условие* — строка критерия. К примеру, следующая инструкция отберет все задачи сотрудника с табельным номером 1 (Андрей Рысь):

```
rstTasks.Filter = 1
```

Если в наборе записей содержатся только поля имени и фамилии, можете воспользоваться следующей конструкцией фильтра:

```
rstTasks.Filter = "LastName = 'Рысь'"
```

или

```
rstTasks.Filter = "LastName = 'Рысь' AND FirstName = 'Андрей'"
```

При использовании свойства `Filter` отфильтрованный набор записей становится текущим курсором. Это значит, что большая часть свойств и методов будет применяться только к отфильтрованным записям, а не ко всему исходному набору записей.

В выражении *условия* можно использовать любые поля, существующие в наборе данных, а также следующие операторы: `>`, `<`, `>=`, `<=`, `<>`, `=` и `LIKE`. Не забывайте обрамлять значения соответствующими ограничителями: кавычками — строковые константы и знаками диеза — константы дат.

Для снятия фильтра используется следующий синтаксис:

```
набор_записей.Filter = adFilterNone
```


Константа `adFilterNone` идентична присвоению свойству строки нулевой длины. В результате снятия фильтра текущая запись смещается к первой в исходном наборе данных.

Предупреждение

В Access и языке VBA, в отличие от библиотеки ADO, поддерживаются символы макроподстановки * и ?. Если для наполнения набора данных вы используете выражение SQL, то должны использовать символы макроподстановки ADO. В этой библиотеке символ звездочки эквивалентен символу процента, а символ вопросительного знака — символу подчеркивания. При использовании символов * и ? ошибка выполнения не возникнет; вместо этого ADO проинтерпретирует использованные символы буквально, результатом чего может стать пустой или заведомо неверный набор данных.

Использование свойства Recordset

Некоторые объекты Access — простой и комбинированный список, а также объекты форм и отчетов — имеют свойство `Recordset`. Результатом является элемент управления, форма или отчет, связанный с набором данных. Это аналогично связыванию объекта с запросом или таблицей.

Процесс работы со свойством `Recordset` прост — создайте этот объект обычным способом, после чего установите свойство объекта следующим образом:

```
Set объект.Recordset = набор_данных
```

В одной из предыдущих глав был представлен пример добавления комбинированного списка в форму `Employees` с целью фильтрации данных. Эта форма была привязана к таблице `Employees` и как таковая отображала данные обо всех сотрудниках. Добавленный комбинированный список позволял временно устанавливать фильтр на данные, касающиеся конкретного сотрудника.

Как всегда, в Access одну и ту же задачу можно выполнить несколькими способами. Для фильтрации данных можно было бы использовать и свойство `Recordset` формы. Разница состоит в том, что в таком случае форма была бы привязана к набору данных, а не к исходной таблице. Если определить набор данных нужного типа, сохранится возможность обновления данных. В дополнение к этому, в форме будут отображаться данные только об одном сотруднике, и в любой момент времени в наборе данных формы будет находиться всего одна запись.

Откройте стандартный модуль (или воспользуйтесь модулем `Chapter16` базы данных `TimeTrack`) и введите следующую процедуру:

```
Sub FilterEmployeeForm(val As Integer)
    Dim rstEmployees As ADODB.Recordset
    DoCmd.OpenForm "Employees"
    Set rstEmployees = New ADODB.Recordset
    With rstEmployees
        .CursorLocation = adUseClient
        .Open "SELECT * " _
```



```

        & "FROM Employees "
        & "WHERE EmployeeID = " & val, _
        CurrentProject.Connection, adOpenStatic, adLockOptimistic
    End With
    Set Forms("Employees").Recordset = rstEmployees
End Sub

```

После этого в окне Immediate введите следующую инструкцию:

```
FilterEmployeeForm(3)
```

Теперь зайдите в только что открытую форму и вы увидите, что в ней отображены данные для сотрудника с табельным номером 3 — Руслана Пилипейко. Обратите внимание, что на панели навигации отображено значение 1. Это указывает на то, что в наборе данных формы содержится всего одна запись.

Попробуйте выбрать из комбинированного списка другого сотрудника. Так как в наборе данных информация о других сотрудниках отсутствует, это приведет к ошибке, сообщение о которой показано на рис. 16.4. Закройте окно сообщения, щелкнув на кнопке **End**. Если вы собираетесь использовать форму таким образом, придется поправить свойства поиска элемента комбинированного списка.

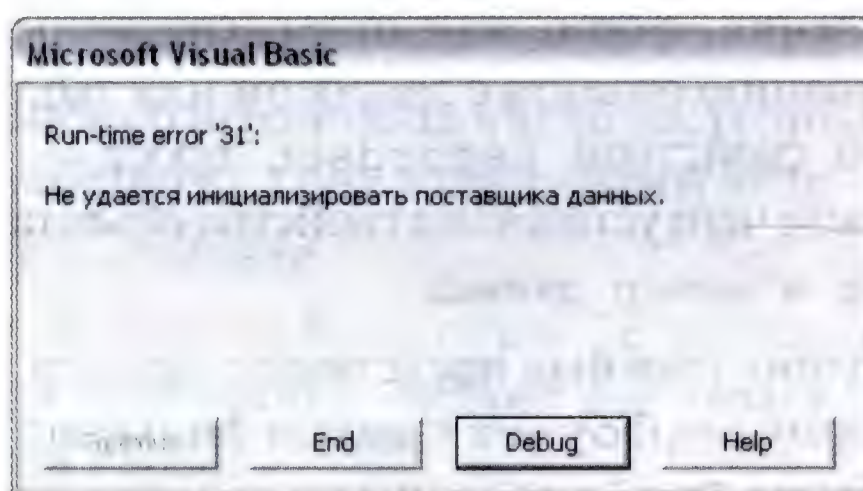


Рис. 16.4. Когда форма открывается с использованием свойства Recordset, элемент комбинированного списка своих функций не выполняет

В описанном примере мы не уничтожали и не закрывали объект Recordset по той причине, что как таковой его можно продолжать использовать. В рабочем же примере вам потребуется вначале закрывать, а затем уничтожать объекты наборов данных после того, как потребность в них отпадает. Чаще всего это выполняют в процедуре закрытия формы.

Предупреждение

Приведенный пример работает по той причине, что форма изначально была подключена к источнику данных и вы не изменяли имен элементов управления. Таким образом, имена элементов продолжали соответствовать именам полей набора данных. Если вы измените имена элементов управления, пример потеряет работоспособность до тех пор, пока вы не сопоставите элементам реальные имена полей набора данных.

ПРАКТИКУМ**Кто подключен к базе данных?**

Предположим, что база данных примеров TimeTrack работает в многопользовательском режиме. Это значит, что администратор базы данных (или ее разработчик) должен периодически открывать ее в эксклюзивном режиме и выполнять общие технологические операции. Однако перед тем, как открыть базу данных в монопольном режиме, администратор должен знать, кто в настоящий момент к ней подключен, чтобы предупредить этих сотрудников о временном закрытии соединения.

Это может быть устное предупреждение или телефонный звонок. А что делать, если сотрудник по каким-либо причинам покинул рабочее место, оставив базу данных открытой? Самое простое решение этой задачи — составить список подключенных к базе данных сотрудников средствами ADO.

Для упрощения процесса на форму переключателей Switchboard мы добавим еще одну командную кнопку. При нажатии на нее будет открываться простая форма, содержащая список имен сотрудников, подключенных к базе данных в настоящее время. Чтобы упростить задачу, не будем нагружать эту форму другими функциями. В то же время, применяя описанную ниже методику к реальной базе данных, можно повысить функциональность этой формы. К примеру, в ответ на нажатие кнопки можно отправлять подключенным пользователям электронные сообщения или распечатать список соответствующих внутренних телефонных номеров.

В этом разделе представлен *набор данных схем*. При обычных условиях объект Recordset содержит в себе записи из некоторой таблицы; набор данных схем содержит информацию о самой таблице или базе данных.

Для начала добавим на форму Switchboard новую командную кнопку. Присвоим ей имя cmdCurrentUsers и соответствующую надпись. Теперь в модуль формы можно ввести следующую процедуру обработки события, связанного с нажатием пользователя на кнопку:

```
Private Sub cmdCurrentUsers_Click()  
On Error GoTo HandleErr  
DoCmd.OpenForm "CurrentUsers"  
ExitHere:  
Exit Sub  
HandleErr:  
MsgBox "Ошибка " & Err.Number & ": " & _  
Err.Description  
Resume ExitHere  
End Sub
```

Не волнуйтесь о том, что форма CurrentUser еще не существует — ее мы создадим немного позже. А пока что сохраните и закройте форму Switchboard.

Создайте новую форму и поместите на нее элемент списка, установив свойство **Тип источника строк** в значение **Список значений**. Установите другие свойства элемента согласно указаниям в табл. 16.7. После этого в модуль формы введите следующую процедуру:


```
Private Sub Form_Load()  
    'Наполнение списка активных сотрудников  
    Dim strUsers As String  
    On Error GoTo HandleErr  
    strUsers = ReturnUsers  
    lstCurrentUsers.RowSource = strUsers  
ExitHere:  
    Exit Sub  
HandleErr:  
    MsgBox "Ошибка " & Err.Number & ": " & _  
        Err.Description  
    Resume ExitHere  
End Sub
```

Сохраните форму под именем CurrentUsers и закройте ее.

Таблица 16.7. Свойства формы CurrentUsers

Свойство	Значение
Полосы прокрутки	Отсутствуют
Кнопки перехода	Нет
Область выделения	Нет

Теперь откройте новый стандартный модуль и введите следующий программный код (можно использовать существующий модуль Chapter16 базы данных примеров):

```
Public Const JET_SCHEMA_USERROSTER =  
    "{947bb102-5d43-11d1-bdbf-00c04fb92675}"  
  
Public Function ReturnUsers() As String  
    Dim cnn As ADODB.Connection  
    Dim rst As ADODB.Recordset  
    Set cnn = New ADODB.Connection  
    On Error GoTo HandleErr  
    'Подключаемся к базе данных  
    Set cnn = CurrentProject.Connection  
    'Открываем схематический набор записей  
    'для захвата метаданных пользователя  
    Set rst = cnn.OpenSchema(adSchemaProviderSpecific, , _  
        JET_SCHEMA_USERROSTER)  
    'Возвращаем текущих пользователей  
    rst.MoveFirst  
    Do Until rst.EOF  
        ReturnUsers = rst(0) & ";" & ReturnUsers  
        rst.MoveNext  
    Loop  
  
ExitHere:  
    rst.Close  
    Set rst = Nothing  
    cnn.Close  
    Set cnn = Nothing  
    Exit Function
```



```

HandleErr:
  MsgBox "Ошибка " & Err.Number & ": " & _
    Err.Description
  Resume ExitHere
End Function

```

Первые строки объявления константы поместите в область общих объявлений модуля. Сохраните модуль и закройте его.

Для того чтобы проверить работоспособность программы, откройте форму Switchboard и щелкните на кнопке **Пользователи**. Откроется новая форма, показанная на рис. 16.5. Как видите, в настоящий момент к базе данных подключен только один пользователь — вы сами (поскольку в форме отображается не имя пользователя, а имя его компьютера).

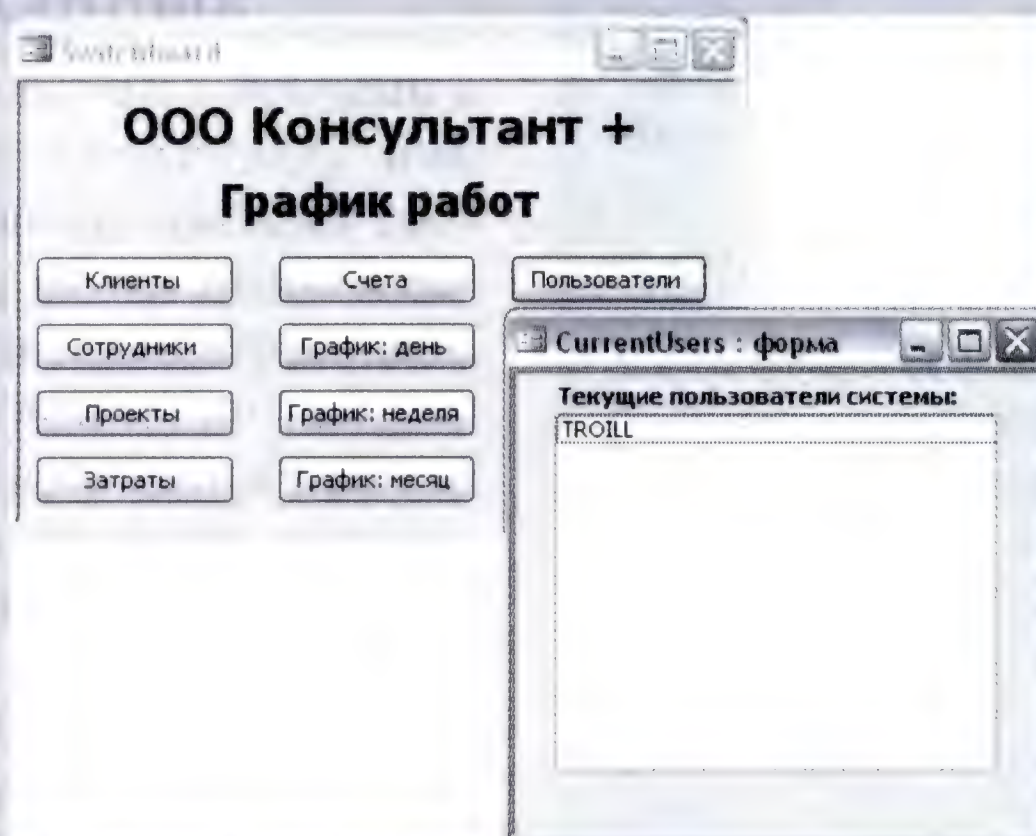


Рис. 16.5. В списке формы перечислены все пользователи, подключенные к базе данных

В принципе, для вызова этой формы совершенно не обязательно вызывать форму Switchboard, однако сведение вызова всех форм воедино, с точки зрения конечного пользователя, лишь добавляет удобства. При щелчке на кнопке **Пользователи** открывается форма CurrentUsers. В ответ на событие открытия формы выполняется процедура ReturnUsers.

Метод OpenSchema сообщает ADO, какой тип информации нам нужен. В данном случае мы использовали константу JET_SCHEMA_USERROSTER. Эта длинная строка, заданная в области общих объявлений, определяет глобальный идентификатор GUID, определяющий схемы текущих пользователей.

После того как процедура ReturnUsers вернет список текущих пользователей в одной строке, эта строка будет использована в качестве источника строк спискового элемента.

Манипуляции с данными средствами ADO

Перемещение по набору данных

После того как данные извлечены с помощью объекта `Recordset`, ими можно манипулировать любыми способами, даже если вашей главной целью был всего лишь просмотр данных. Многие из того, с чем вы познакомитесь в этой главе, пригодятся при поиске, добавлении, удалении и редактировании существующих данных.

Перемещение по набору данных требует установки *указателя текущей записи* внутри курсора (извлеченных данных). Другими словами, вы указываете на некоторую запись в наборе, что делает ее указателем текущей записи. Для перемещения по объекту `Recordset` используют пять следующих методов:

- `MoveFirst`. Перемещение указателя к первой записи в наборе.
- `MoveNext`. Перемещение указателя к следующей записи в наборе.
- `MovePrevious`. Перемещение указателя к предыдущей записи в наборе.

17

В ЭТОЙ ГЛАВЕ

Перемещение по набору данных.....	295
Поиск данных в объекте <code>Recordset</code>	299
Добавление данных в объекте <code>Recordset</code>	303
Удаление данных в объекте <code>Recordset</code>	305
Обновление данных в объекте <code>Recordset</code>	306
Использование транзакций для изменения группы записей	309

- `MoveLast`. Перемещение указателя к последней записи в наборе.
- `Move`. Перемещение указателя на заданное в аргументе число записей.

Последний метод требует некоторых пояснений. Он поддерживает два аргумента и имеет следующий синтаксис:

```
Move x[, начало]
```

где `x` — значение типа `Long`, определяющее количество записей, на которые требуется сместиться, а `начало` — значение типа `String` или `Variant`, служащее своеобразной закладкой. Этот аргумент может принимать значения констант, перечисленных в табл. 17.1.

Таблица 17.1. Константы закладок

Константа	Значение	Описание
<code>adBookmarkCurrent</code>	0	Начиная с текущей записи
<code>adBookmarkFirst</code>	1	Начиная с первой записи
<code>adBookmarkLast</code>	2	Начиная с последней записи

Если число `x` больше нуля, указатель смещается вперед, в противном случае — назад. Указание отрицательного значения `x` является единственным способом смещения назад в наборе данных с последовательным доступом. В то же время этот метод непредсказуем, так как не все поставщики поддерживают данную возможность. Если аргумент `начало` опустить, перемещение будет произведено относительно текущей записи.

Вы никуда не сможете переместиться в наборе данных, если случайно застряли перед первой или после последней записи. При перемещении по записям постоянно нужно проверять эти два места, используя булевы свойства `BOF` и `EOF`. Свойство `BOF` (`beginning of file`) принимает значение `True`, когда указатель перемещается в позицию перед первой записью набора. Аналогично, свойство `EOF` равно `True`, когда указатель перемещается в позицию после последней записи.

Чтобы продемонстрировать, насколько важными являются эти два свойства, давайте посмотрим, что случится, если их игнорировать. В приведенной ниже процедуре из таблицы `Clients` извлекаются данные, после чего названия клиентов выводятся в окно `Immediate`.

```
Sub PrintClients()  
    'Печать названий клиентов в окно Immediate  
    Dim rst As ADODB.Recordset  
    Set rst = New ADODB.Recordset  
    rst.Open "Clients", CurrentProject.Connection  
    rst.MoveFirst  
    Do  
        Debug.Print rst(1)  
        rst.MoveNext  
    Loop
```



```

rst.Close
Set rst = Nothing
End Sub

```

Введите эту процедуру в стандартный модуль или воспользуйтесь готовым модулем Chapter17 в базе данных TimeTrack. Поместите курсор внутри тела процедуры и нажмите клавишу <F5>.

Названия клиентов будут в цикле выводиться в окно Immediate, однако после того, как будет предпринята попытка перемещения за последнюю запись, интерпретатор выдаст сообщение об ошибке (рис. 17.1), так как в данном случае печатать нечего. Щелкните на кнопке End и закройте окно сообщения.

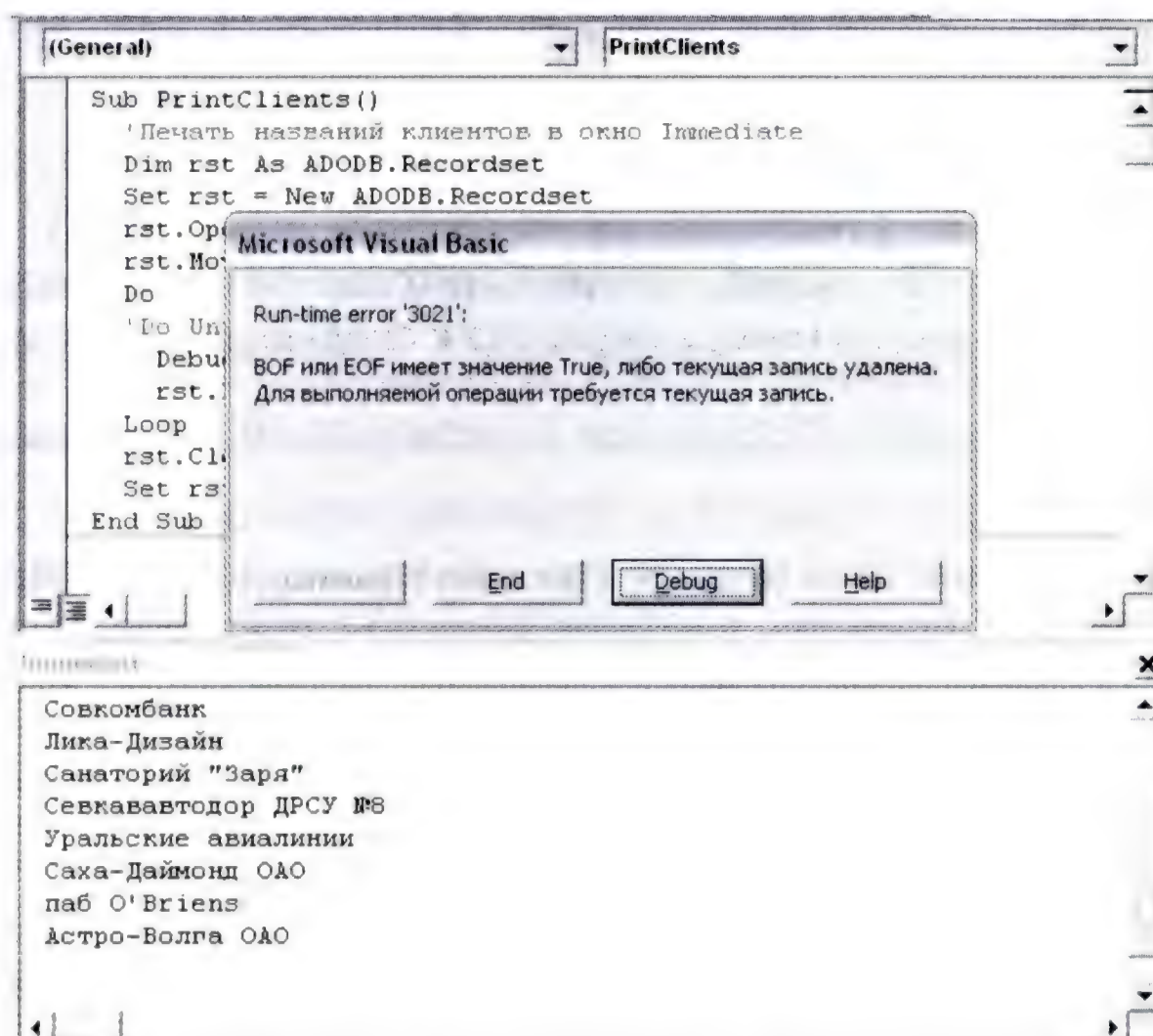


Рис. 17.1. Если вы будете отслеживать место нахождения указателя текущей записи, то сможете избежать подобных ошибок

В данном случае решение крайне просто — нужно заменить оператор Do на следующую конструкцию:

```
Do Until rst.EOF
```

Новый оператор Do Until проверяет истинность свойства EOF, и как только оно станет истинным, прерывает выполнение цикла. (Это свойство истинно в том случае, когда указатель текущей записи находится в конце файла.)

Как и какие из этих двух свойств проверять, зависит исключительно от поставленной задачи и методов ее реализации. Можно заключить фрагмент программы в цикл Do Until, как в приведенном примере, или использовать оператор If. К тому же иногда может потребоваться проверка обоих свойств — как конца, так и начала файла. В данном случае это не понадобилось, поскольку цикл прерывается только в конце файла.

Совет

Свойства EOF и BOF можно также использовать для проверки пустоты набора записей. Если набор записей пуст, то оба эти свойства одновременно равны значению True. К примеру, комбинированное условие `rst.EOF AND rst.BOF` будет истинным только в случае, когда набор записей `rst` пуст.

Обратите внимание, что в приведенном примере использованы два метода, о которых мы говорили ранее: `MoveFirst` и `MoveNext`. Применение первого метода до начала цикла гарантирует установку указателя в первую запись, так что нет никакой необходимости проверять свойство `BOF`. В теле же цикла используется метод `MoveNext` для смещения указателя к следующей записи набора данных, после чего выполнение цикла продолжается.

Предупреждение

Если вы случайно написали цикл, который никогда не завершается, проверьте, был ли вставлен в него вызов метода перехода к следующей записи.

Предупреждение

При написании операций перемещения по набору записей всегда учитывайте тип объекта `Recordset`. К примеру, в наборе записей с последовательным доступом вы не сможете применять метод `MovePrevious`. Более подробно типы курсоров рассматривались в соответствующем разделе главы 16.

Ссылка на поля набора данных

Объект `Recordset` содержит коллекцию `Fields`, в которой находится по одному объекту `Field` для каждого поля данных. Существует масса способов обращения к данным. Например, можно использовать объект `Field` следующим образом:

```
набор_данных.Fields(имя_поля).Value
```

где `набор_данных` представляет объект `Recordset`, а `имя_поля` — название поля набора данных. Инструкция, приведенная в качестве примера, возвращает данные из поля `Client` текущей записи:

```
rst.Fields("Client").Value
```

В качестве альтернативы можно использовать индексное значение поля. Если точно известно, что столбец `Client` является вторым в наборе записей по счету, к тому же результату приведет и следующее обращение:

```
rst(1)
```

При использовании индексных значений помните, что индекс самого левого столбца равен нулю. Следовательно, индексом второго столбца будет единица, третьего — двойка и т.д.

Совет

Нет ничего зазорного в использовании индексов для обращения к значениям набора данных, однако следует учесть два вопроса. Во-первых, индексное значение поля может изменяться при изменении структуры источника данных. К примеру, вы можете добавить или удалить некоторый столбец, в результате чего их нумерация сместится. К тому же несмотря на некоторое замедление процесса при использовании для обращения метода `Fields`, читабельность программы значительно повышается.

Поиск данных в объекте `Recordset`

Часто в работе требуется искать записи по определенному критерию, а не просто перемещаться между ними вперед и назад. Метод `Find` библиотеки ADO гибок и эффективен, однако он не позволяет объединять несколько условий оператором `AND`. Метод `Find` имеет следующий синтаксис:

`набор_записей.Find критерий [, пропуски] [, направление] [, начало]`

Только первый из этих аргументов является обязательным:

- *критерий*. Строковое значение в форме предложения `WHERE` языка SQL, однако без ключевого слова `WHERE`.
- *пропуски*. Целочисленное значение, которое определяет, через сколько строк от текущей следует начинать поиск. Если этот аргумент при вызове опущен, поиск начинается с текущей строки.
- *направление*. Определяет направление поиска: вперед (константа `adSearchForward`) или назад (константа `adSearchBackward`).
- *начало*. Значение типа `Variant`, определяющее место начала поиска. Если аргумент опущен, поиск начинается с текущей записи. В качестве начальной точки поиска можно использовать три константы: `adBookmarkCurrent` (с текущей строки, используется по умолчанию), `adBookmarkFirst` (с первой строки) или `adBookmarkLast` (с последней строки).

Совет

Когда поиск основан на нескольких критериях, используйте свойство `Filter` в формате `набор_записей.Filter = "условие1 AND условие2"`. Более подробно это свойство рассматривалось в главе 16.

Метод `Find` позволяет найти запись на основе строки поиска. Строка поиска задается в форме предложения `WHERE` языка SQL, в котором опущено ключевое слово `WHERE`:

`набор_записей.Find "поле = значение"`

В следующей процедуре проиллюстрирован метод `Find`.


```

Sub FindClients(str As String)
    'Принимает строковое значение в качестве критерия поиска
    'для метода Find
    Dim rst As ADODB.Recordset
    Dim strSearch As String
    strSearch = "City = '" & str & "'"
    Set rst = New ADODB.Recordset
    With rst
        .Open "Clients", CurrentProject.Connection, adOpenStatic, _
            adLockPessimistic
        .MoveFirst
        .Find strSearch
        Do Until .EOF
            Debug.Print rst("Client")
            .Find strSearch, 1
        Loop
    End With
    rst.Close
    Set rst = Nothing
End Sub

```

Можно ввести эту процедуру в стандартный модуль или использовать готовую процедуру из модуля Chapter17 базы данных примеров. После этого введите в окне **Immediate** следующую инструкцию:

```
FindClients("Москва")
```

На рис. 17.2 показан результат выполнения процедуры — список клиентов, расположенных в Москве.



Рис. 17.2. Поиск данных с помощью метода Find

После создания и наполнения объекта набора данных с помощью метода Find мы находим первую запись, в которой в поле City (Город) содержится значение аргумента str. В данном случае этому аргументу мы передали строку "Москва". В цикле Do выводится на печать название найденной компании, после чего снова выполняется поиск с теми же параметрами. Обратите внимание, что, используя аргумент *пропуски*, мы избегаем дополнительного смещения записи с помощью метода MoveNext. Если бы это смещение не проводилось, получился бы бесконечный цикл, в котором постоянно бы печаталось название одной и той же компании.

В методе Find можно использовать операторы отличные от знака равенства. К примеру, можно использовать критерий City > 'Москва' и найти в данных первый город после Москвы.

Альтернатива в поиске — метод Seek

При определенных условиях метод Seek может оказаться предпочтительнее метода Find, поскольку он производит поиск быстрее. Однако для того чтобы набор записей поддерживал метод Seek, необходимо выполнение двух условий:

- Тип набора записей должен быть прямой таблицей, открытой с параметром `adCmdTableDirect`. Этот параметр специфичен для баз данных Microsoft Access (вы ведь знаете, что с помощью библиотеки ADO можно работать и с другими типами данных).
- Свойство `Index` объекта набора записей должно быть установлено в индексированное поле, по которому будет проводиться поиск.

Примечание

Чтобы использовать метод Seek, нужно знать название индекса по поисковому полю, существующему в таблице. В большинстве случаев это особых проблем не вызывает, так как при создании индекса вы сами, вручную или программным образом, присваиваете ему имя. Если вы забыли задать имя индекса в программе явно, Access скомбинирует имена полей, участвующих в индексе, и сформирует нужное имя индекса. По-другому дело обстоит с полями первичных ключей. Access автоматически присваивает им имя *PrimaryKey* (если первичный ключ вы создаете программным путем, это имя вы можете заменить другим).

Если вы точно не знаете имена индексов, можете узнать их программным путем. Для этого вам потребуется библиотека *ADOX*, о которой более подробно мы будем говорить в главе 18. Однако, забегаая вперед, следует отметить, что в окне **Immediate** список индексов можно получить с помощью следующей процедуры:

```
Sub FindIndex (tblName As String)
    'Поиск всех индексов в таблице
    Dim cat As ADOX.Catalog
    Dim tbl as ADOX.Table
    Dim idx As ADOX.Index
    Set cat = New ADOX.Catalog
    Set cat.ActiveConnection = CurrentProject.Connection
    Set tbl = cat.Tables(tblName)
    For Each idx In tbl.Indexes
        Debug.Print idx.Name
    Next idx
End Sub
```

Предупреждение

При использовании метода Seek помните, что поля типов Memo, Hyperlink и OLE Object проиндексировать нельзя.

Метод Seek использует следующий синтаксис:

набор_записей.Seek индекс, параметр

где *параметром* может выступать любая из констант, перечисленных в табл. 17.2. Для того чтобы понять смысл этих констант, нужно понять процесс

работы метода `Seek`. Этот метод не ищет точное значение в поле (что называют методом *табличного сканирования*), как это выполняется приложением Access для неиндексированных полей.

Внутренняя работа индексов достаточно сложна, однако для решения поставленных задач достаточно представить их как отсортированный классификатор, хранящий ключевые значения и указатели на вхождения в индексированное поле.

Таблица 17.2. Встроенные константы параметра метода `Seek`

Константа	Описание
<code>adSeekAfterEQ</code>	Выбирает индексированное значение, равное заданному. Если подходящего значения не найдено, останавливается на ключевом значении, непосредственно следующем за искомым (что в принципе возможно только в случае сортировки внутренних значений)
<code>adSeekAfter</code>	Выбирает первое значение индексного ключа, равного заданному после текущей точки
<code>adSeekBefore</code>	Выбирает первое значение индексного ключа, равное заданному до текущей точки
<code>adSeekBeforeEQ</code>	Выбирает значение индексного ключа, равного искомому. Если подходящего значения не найдено, останавливается на ключевом значении, непосредственно предшествующем искомому
<code>adSeekLastEQ</code>	Выбирает последний индексный ключ, равный искомому

Индексом является всего лишь список значений с указателями на фактические строки, в которых хранятся данные. Так как эти значения отсортированы (внутренне), Access может достаточно быстро определить, где искомое значение может находиться. При этом метод `Seek` ищет ключевые значения, равные искомому, не сканируя всю таблицу.

В представленной ниже процедуре используется метод `Seek` для поиска конкретного клиента в таблице `Clients` базы данных `TimeTrack`.

```
Sub SeekClient(ClientID As Long)
    'Поиск записи клиента
    'с помощью метода Seek
    Dim rst As New ADODB.Recordset
    With rst
        .Open "Clients", CurrentProject.Connection, _
            adOpenDynamic, adLockOptimistic, adCmdTableDirect
        .Index = "PrimaryKey"
        .Seek ClientID, adSeekAfterEQ
    End With
    If rst.EOF Then
        MsgBox "Для клиента " & _
            & ClientID & "записи не найдены", vbOKOnly
    Else
        MsgBox "Клиентом " & ClientID & "является " & rst("Client") _
```



```

        & "", vbOKOnly
    End If
    rst.Close
    Set rst = Nothing
End Sub

```

Процедуру можно либо ввести в стандартный модуль, либо воспользоваться готовой, содержащейся в модуле процедурой Chapter17. Теперь в окне Immediate введите следующую инструкцию:

```
SeekClient(1)
```

После создания и наполнения прямого табличного набора записей свойство Index устанавливается в индекс первичного ключа. После этого используется метод Seek для поиска первой записи со значением 1 в поле ClientID (это — поле первичного ключа). Далее в конструкции If открывается окно, отображающее название найденного клиента (рис. 17.3). Если же вхождений искомого значения не найдено, открывается окно с соответствующим сообщением. В любом случае для закрытия окна нужно щелкнуть на кнопке ОК.

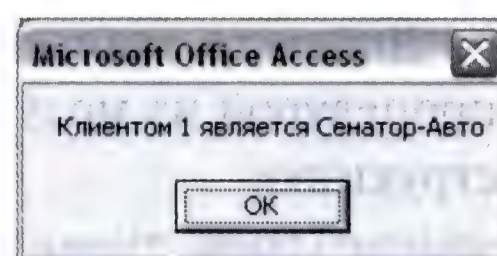


Рис. 17.3. Данные, найденные методом Seek, можно отобразить в окне

Добавление данных в объекте Recordset

Иногда может потребоваться добавить некоторые данные с помощью объекта набора данных Recordset. Этот процесс немного сложнее, чем просмотр или поиск данных. Добавление новой записи требует выполнения трех операций:

- вызова метода AddNew библиотеки ADO для создания в наборе данных новой записи;
- присвоения значений всем необходимым полям;
- вызова метода Update для сохранения новой строки в источнике данных. Если не выполнить метод Update, все новые данные исчезнут без предупреждения.

Совет

Объект Recordset является не единственным средством добавления новых записей. С помощью объекта Command можно выполнить выражение на языке SQL непосредственно в источнике данных. Этот объект подробно описан в главе 16.

Рассмотрим простой пример, в котором в таблицу сотрудников Employees базы данных TimeTrack добавляется новая запись.

```

Sub AddEmployee(fname As String, lname As String)
    'Добавление новой записи о сотруднике
    Dim rst As ADODB.Recordset
    Set rst = New ADODB.Recordset

```



```
With rst
    .Open "Employees", CurrentProject.Connection, adOpenDynamic, _
        adLockPessimistic
    .AddNew
    .Fields("FirstName") = fname
    .Fields("LastName") = lname
    .Update
End With
rst.Close
Set rst = Nothing
End Sub
```

Добавьте процедуру в стандартный модуль или воспользуйтесь готовой процедурой из модуля Chapter17. В окне Immediate введите следующую инструкцию:

```
AddEmployee "Иван", "Демидов"
```

После этого откройте таблицу Employees, и вы обнаружите в ней новую запись (рис. 17.4).

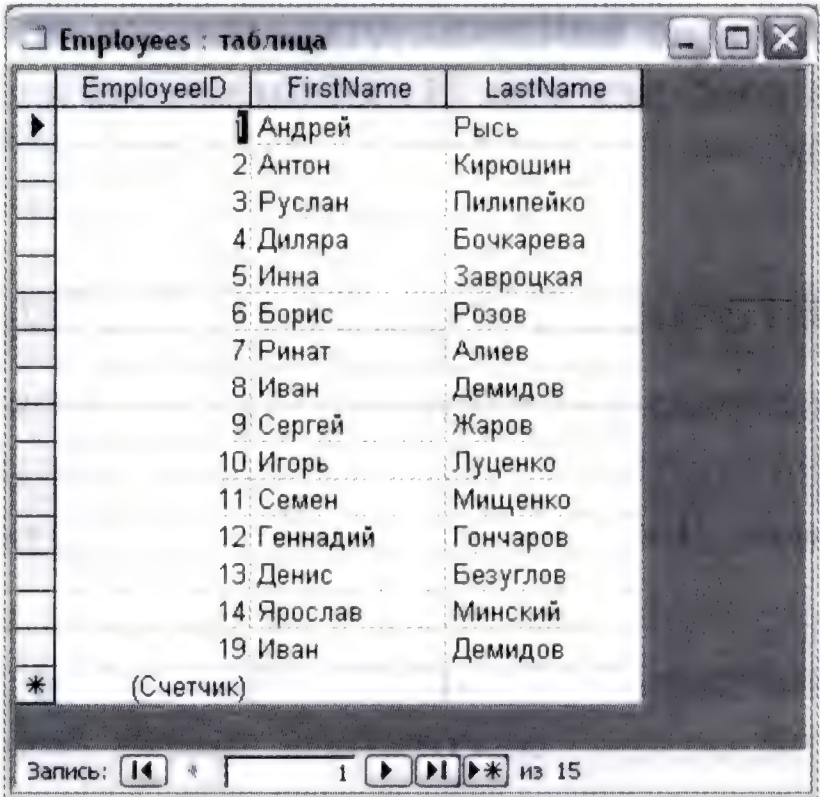


Рис. 17.4. Для добавления новых записей можно использовать объект Recordset

После открытия и заполнения объекта Recordset с помощью метода AddNew мы высвобождаем место для новой записи. Затем, используя две инструкции, в поля новой записи мы заносим значения, переданные в аргументах. В заключение применяется метод Update, обновляющий данные в источнике. Метод Update имеет следующий синтаксис:

```
набор_записей.Update [поля] [, значения]
```

где аргумент *поля* идентифицирует обновляемые поля, а *значения* заменяют существующие значения.

Вам может показаться странным, почему мы не добавляли значения в поле EmployeeID. Все дело в том, что это поле имеет тип AutoNumber (Автоматическая нумерация), и при добавлении новой записи оно заполняет-

ся приложением Access автоматически. Попытка ручного присвоения значения этому полю приведет к ошибке.

При добавлении новых данных нужно знать источник данных. Попытка добавления неверного типа данных приведет к ошибке выполнения. Следует отметить, что в приведенном примере не была заложена обработка ошибок. При вводе новых записей программисты обычно добавляют проверку значений на самые распространенные типы ошибок. К примеру, можно проверить, не является ли новая запись полным двойником уже существующей, а также не противоречат ли данные установленным правилам.

Удаление данных в объекте Recordset

Удаление записей сводится к поиску нужной строки и применением на ней метода Delete объекта Recordset. Будьте внимательны, так как после выполнения операции удаления данные исчезают бесследно. Учтите, что текущая запись остается текущей даже после своего удаления. Для полного подтверждения операции удаления после выполнения метода Delete следует установить указатель текущей записи на реальную строку с помощью одного из методов навигации или поиска.

Синтаксис метода Delete крайне прост:

```
набор_записей.Delete
```

В первую очередь нужно выбрать запись, подлежащую удалению, а после этого уже вызывать метод Delete. С помощью приведенной ниже процедуры мы удалим новую запись в таблице сотрудников Employee, которая была добавлена в предыдущем разделе.

```
Sub DeleteEmployee(emID As Long)
    'Удаление записи о сотруднике
    Dim rst As ADODB.Recordset
    Dim strSearch As String
    Dim strName As String
    Dim bytResponse As Byte
    strSearch = "EmployeeID = "& emID
    Set rst = New ADODB.Recordset
    With rst
        .Open "Employees", CurrentProject.Connection, adOpenDynamic, _
            adLockPessimistic
        .Find strSearch
        If .EOF = True Then
            MsgBox "Не существует записи для сотрудника " & _
                & emID, vbOKOnly
            Exit Sub
        Else
            strName = rst.Fields("FirstName") & _
                "" & rst.Fields("LastName")
            bytResponse = MsgBox("Вы действительно хотите удалить " & _
                запись об " & _
                & strName, vbYesNo)
            If bytResponse = vbYes Then
```



```
        .Delete
    End If
End If
End With
rst.Close
Set rst = Nothing
End Sub
```

Добавьте процедуру в стандартный модуль или воспользуйтесь готовой процедурой из модуля Chapter17. В окне Immediate введите следующую инструкцию:

```
DeleteEmployee 19
```

(В качестве аргумента процедуры используйте тот номер, под которым в примере предыдущей главы был добавлен новый сотрудник.) После создания и наполнения объекта Recordset с помощью метода Find мы находим запись с нужным значением поля EmployeeID. Далее отображается запрос подтверждения операции удаления записи (рис. 17.5). Если щелкнуть на кнопке Да, будет выполнен метод Delete, если на кнопке Нет — метод Delete будет опущен. Если в таблице не будет найдено поле с заданным идентификатором, откроется окно с соответствующим сообщением. Щелкните в нем на кнопке ОК.

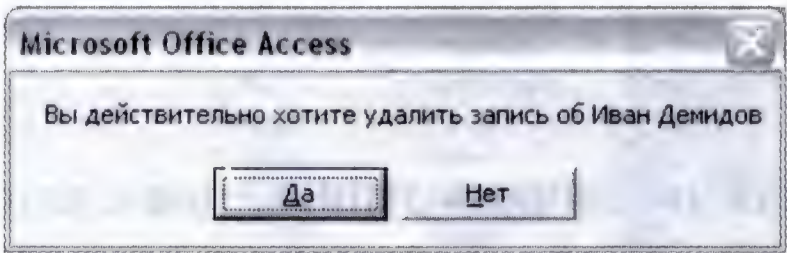


Рис. 17.5. Перед удалением записи всегда лучше получить дополнительное подтверждение пользователя

Обновление данных в объекте Recordset

Обновление (или редактирование) существующих данных в объекте Recordset сводит воедино многие методики, описанные в предыдущих разделах. После создания и наполнения данными объекта Recordset следует отыскать записи, которые требуется обновить. После этого нужно подтвердить изменения с помощью метода Update, как было сделано при добавлении данных.

Предупреждение

Перед тем как физически выполнять любую из процедур настоящего раздела, измените размер поля State (Регион) таблицы Clients до 11 символов или больше. Если этого не сделать, выполнение процедуры может привести к ошибке при попытке изменения поля региона, к примеру, на Центральный.

В первую очередь посмотрим на явный вызов метода Update. Это — самый безопасный способ внесения изменений в источник данных, поскольку он открывает дополнительный уровень управления процессом. В приведен-

ном ниже примере мы будем изменять регион клиентов, расположенных в Москве, с общего определения Россия на реальное название территориального региона — Центральный (имеется в виду федеральный округ). Эта процедура принимает два значения: название города и новое название региона.

```
Sub ChangeStateExplicit1(st As String, State As String)
    'Явный вызов метода Update для подтверждения изменений
    Dim rst As ADODB.Recordset
    Set rst = New ADODB.Recordset
    With rst
        .Open "Clients", CurrentProject.Connection, adOpenDynamic, _
            adLockOptimistic
        .Find "City = '" & st & "'"
        If .EOF = True Then
            MsgBox "Вхождение не найдено", vbOKOnly
            Exit Sub
        Else
            .Update "State", State
            MsgBox .Fields("State"), vbOKOnly
        End If
    End With
    rst.Close
    Set rst = Nothing
End Sub
```

Введите данную процедуру в стандартный модуль или воспользуйтесь готовой процедурой из модуля Chapter17. После этого в окне Immediate выполните следующую инструкцию:

```
ChangeStateExplicit1 "Москва", "Центральный"
```

Когда откроется окно сообщения с новым названием региона (рис. 17.6), щелкните на кнопке ОК. Теперь в первой строке таблицы Clients название региона заменено на Центральный — это первое вхождение в записи города Москва (рис. 17.7).

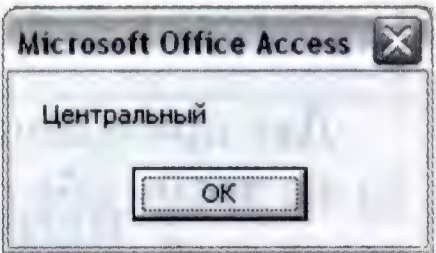


Рис. 17.6. Процедура обновления данных отображает новое значение текущей записи

Clients : таблица							
	ClientID	Client	Address	City	State	Zip	P
+	1	Сенатор-Авто	Ленинградский пр. 37,	Москва	Центральный	125315	095-14
+	2	Интеравтосервис ОАО	ш. Энтузиастов 1А	Балашиха	Россия	143900	095-51
+	3	Комунибанк	Головин М. пер. 12	Москва	Россия	103045	095-91
+	4	Совкомбанк	Гиляровского 57 стр.1	Москва	Россия	107996	095-71
+	5	Лица-Дизайн	Антонова-Овсиенко 15	Москва	Россия	123317	095-21
+	6	Санаторий "Заря"	пос. Черноречье	Грозный	Чечня	364050	8712-1
+	7	Севкававтодор ДРСУ №8	Сальмана 49	Дербент	Дагестан	368600	87240
+	8	Уральские авиалинии	Спутников 6, оф. 101	Екатеринбург	Россия	620910	343-21
+	9	Саха-Даймонд ОАО	Кулаковского 28	Якутск	Саха	677007	4112-1
+	10	паб O'Briens	Михайловская 11	Киев	Украина	01001	044-21
+	11	Астро-Волга ОАО	8 Марта 66, оф. 16	Екатеринбург	Россия	620063	343-21
*	(Счетчик)						

Рис. 17.7. Первое вхождение региона Россия (соответствующее городу Москва) изменено на Центральный

Метод Find обнаруживает первую запись, содержащую значение st в поле City. После этого в найденной записи значение поля State изменяется на значение второго аргумента процедуры.

В качестве альтернативы можно явно присвоить новое значение свойству поля, после чего выполнить метод Update без параметров:

```
Sub ChangeStateExplicit2(st As String, State As String)
    'Явный вызов метода Update для подтверждения изменений
    Dim rst As ADODB.Recordset
    Set rst = New ADODB.Recordset
    With rst
        .Open "Clients", CurrentProject.Connection, adOpenDynamic, _
            adLockOptimistic
        .Find "City = '" & st & "'"
        If .EOF = True Then
            MsgBox "Вхождение не найдено", vbOKOnly
            Exit Sub
        Else
            .Fields("State") = state
            .Update
            MsgBox .Fields("State"), vbOKOnly
        End If
    End With
    rst.Close
    Set rst = Nothing
End Sub
```

Две приведенные выше процедуры имеют всего одно небольшое отличие, в то же время обе они используют явное обращение к методу Update и обе выводят сообщение, информирующее пользователя об изменении данных.

Во второй процедуре можно опустить вызов метода Update — при этом он будет вызван неявно. В тексте процедуры, выполнив присвоение нового значения свойству поля, замените вызов метода Update на какой-нибудь метод перемещения к другой записи (в приведенном примере мы смещаемся вперед и назад):

```
Sub ChangeStateImplicit(st As String, State As String)
    'Неявный вызов метода Update для подтверждения изменений
    Dim rst As ADODB.Recordset
    Set rst = New ADODB.Recordset
    With rst
        .Open "Clients", CurrentProject.Connection, adOpenDynamic, _
            adLockOptimistic
        .Find "City = '" & st & "'"
        If .EOF = True Then
            MsgBox "Вхождение не найдено", vbOKOnly
            Exit Sub
        Else
            .Fields("State") = state
            .MoveNext
            .MovePrevious
            MsgBox .Fields("State"), vbOKOnly
        End If
    End With
```



```
rst.Close  
Set rst = Nothing  
End Sub
```

Обратите внимание, что в последнем примере отсутствует явный вызов метода `Update`. После присвоения нового значения полю в инструкции `.Fields("State") = state` метод `MoveNext` автоматически вызывает метод обновления. После того как изменения сделаны, существует два способа их отмены:

- вызов метода `CancelUpdate`;
- закрытие или уничтожение объекта `Recordset` (установкой его значения в `Nothing`).

Примечание

При создании набора данных для обновления учтите, что при многопользовательском режиме работы с базой данных существенную роль играет тип блокировки записей. Во всех приведенных в данном разделе примерах мы устанавливали оптимистическую блокировку. Это значит, что ADO не будет пытаться разрешить конфликты в тот момент, когда сразу несколько пользователей решат одновременно отредактировать одни и те же данные. Обновить запись не удастся, если другой пользователь обновит ее чуть раньше. Для того чтобы библиотека ADO разрешала конфликты и требовала подтверждения внесения изменений, установите пессимистическую блокировку записей.

Использование транзакций для изменения группы записей

Напрямую Access не поддерживает обработку транзакций. В то же время объект `Connection` библиотеки ADO ее поддерживает, так что у вас есть все шансы использовать достоинства этого метода работы с данными.

Обработка транзакций предполагает объединение обновления нескольких записей в единый процесс. Если одно из обновлений не может быть выполнено (например, в результате ошибки или блокировки записи другим пользователем), не будет выполнена и вся группа. Это значит, что либо вся группа инструкций в транзакции выполняется, либо не выполняется ни одна группа. Этот тип управления процессом очень важен в производственной среде, где изменение одного значения может существенно повлиять на результаты работы. Для примера предположим, что мы принимаем заказ, обновляем номенклатурные номера товаров и затем выставляем коммерческое предложение заказчику. Что случится, если заказчик позвонит и отменит часть сделанного заказа (или даже весь заказ)? Нужно будет не только списать сумму отмены с учетной записи данного клиента, но также вернуть товары на склад. Если сделать одну операцию и не сделать другую, нарушится общий баланс, а поиск такой ошибки отнимет массу времени.

Обработка транзакций не только защищает целостность данных от логических ошибок (как в приведенном выше сценарии) — она способна защитить данные в случае внезапных отказов локальной сети или системы электроснабжения. Когда дело касается Access, в таких ситуациях ничего поделать нельзя. Обработка транзакций в ADO не способна полностью обеспечить безопасность процессов, однако “что-то” все же лучше, чем “ничего”. (Технические аспекты, которые здесь играют роль, выходят за рамки данной книги.)

Для реализации обработки транзакций используются три метода объекта Connection:

- **BeginTrans.** Этот метод отмечает точку, в которой поставщик начинает группу операций изменения данных.
- **CommitTrans.** Этот метод следует выполнить для подтверждения изменений, проведенных группой инструкций.
- **RollbackTrans.** Этот метод отменяет все неподтвержденные изменения.

В предыдущем разделе мы использовали метод Update для изменения названия региона; при этом вызов процедуры приводил к обновлению только одной записи. А теперь вставим в эту процедуру цикл, в котором мы заменим все вхождения данного региона в таблице, однако оформим весь процесс как единую транзакцию, чтобы выполнить изменения во всей таблице либо не выполнить их совсем. Введите следующую процедуру в стандартный модуль или используйте заготовку в модуле Chapter17 базы данных примеров.

```
Sub ChangeStateTransaction(st As String, State As String)
    'Оформление обновления как единой транзакции
    'для подтверждения всех изменений или не подтверждения
    'ни одного
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim bytResponse As Byte
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    With rst
        .Open "Clients", cnn, adOpenDynamic, _
            adLockOptimistic
        .Find "City = '" & st & "'"
        cnn.BeginTrans
        Do Until .EOF
            .Fields("State") = State
            .MoveNext
            .Find "City = '" & st & "'"
        Loop
        bytResponse = MsgBox("Вы подтверждаете изменения " & _
            & "в поле региона?", vbYesNo)
        If bytResponse = vbYes Then
            cnn.CommitTrans
        ElseIf bytResponse = vbNo Then
            cnn.RollbackTrans
        End If
    End With
End Sub
```



```
rst.Close  
Set rst = Nothing  
End Sub
```

(Если вы еще не увеличили длину поля State как минимум до 11 символов, о чем было сказано в предыдущем разделе, сделайте это сейчас.)

В окне Immediate выполните следующую инструкцию:

```
ChangeStateTransaction "Москва", "Центральный"
```

После создания и наполнения объекта Recordset, в процедуре выбирается первая строка с городом Москва. Сразу после этого откройте транзакцию методом BeginTrans. В цикле Do замените значения поля State, после чего переходите к следующей записи и выполните следующий поиск (и так до достижения конца файла). При этом все выполненные изменения значений полей пока не подтверждайте.

Когда откроется окно подтверждения преобразований, в первый раз щелкните на кнопке Нет (рис. 17.8). После этого откройте таблицу Clients и убедитесь, что регионы остались неизмененными. Теперь запустите процедуру еще раз, подтвердив выполнение изменений. На этот раз все значения поля региона в записях, относящихся к Москве, будут изменены. (Если в таблице не будет найдено значений первого аргумента в поле City, в таблице ничего не изменится, однако и предупреждения об этом отображено не будет.)

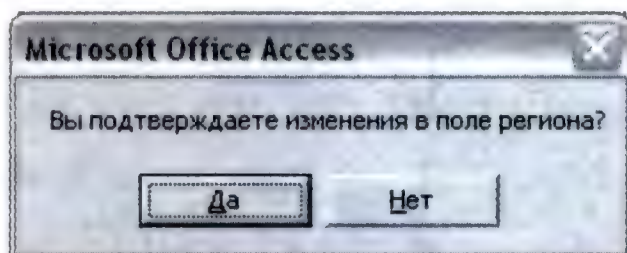


Рис. 17.8. Щелкните на кнопке Нет, чтобы отменить обновления

ПРАКТИКУМ

Использование объекта Recordset для добавления позиций в комбинированный список

В главе 12 мы уже работали с элементом управления комбинированного списка и научились добавлять в его список новые значения. Приведенные примеры работали с источником данных напрямую с помощью выражений SQL. В этом разделе мы выполним ту же задачу, но уже с помощью объекта Recordset.

В настоящее время элемент имени проекта в форме Projects17 (рис. 17.9) имеет вид текстового поля, а это значит, что пользователю необходимо каждый раз вводить в него информацию. Такая конструкция формы крайне неэффективна и потенциально подвержена опечаткам, которые могут нарушить целостность данных. Для ограничения количества возможных ошибок мы заменим текстовое поле элементом комбинированного списка. В этом списке будут содержаться все возможные значения, допустимые в данном поле.

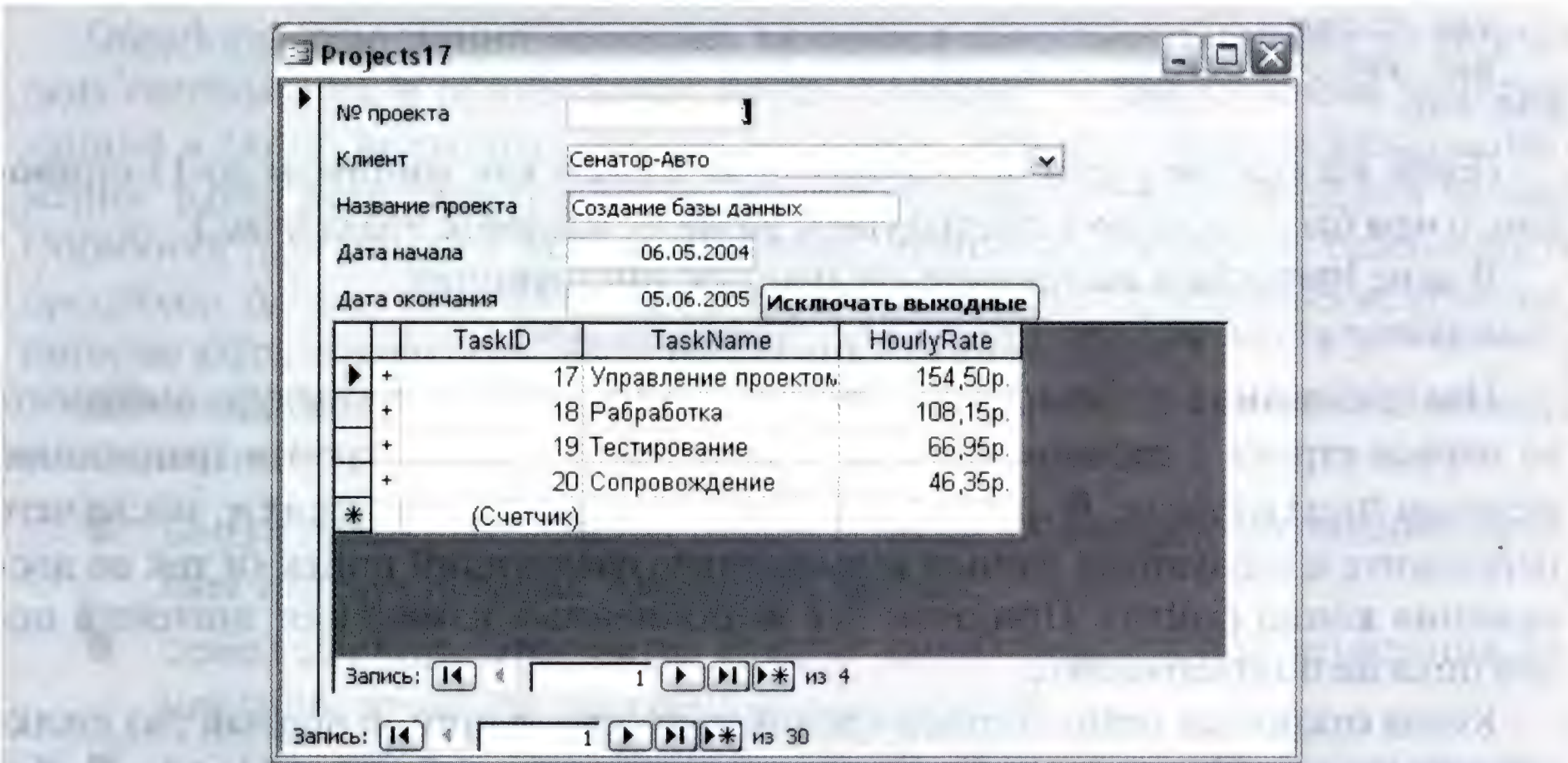


Рис. 17.9. Изменение текстового поля имени проекта на элемент комбинированного списка

Естественно, иногда пользователю потребуется вводить совсем новое название проекта. Как вы уже знаете, комбинированный список способен принять новые значения. Поместим на форму комбинированный список, уже содержащий названия существующих проектов, и способный принять в себя новые. Для этого выполните следующие действия:

- 1. Откройте форму Projects17 в режиме конструктора.
- 2. Щелкните правой кнопкой на текстовом поле имени проекта и выберите в контекстном меню пункт **Преобразовать элемент в⇒Поле со списком (Change To⇒Combo Box)**.
- 3. Установите свойства этого элемента в значения, приведенные в табл. 17.3.

Таблица 17.3. Свойства комбинированного списка

Свойство	Значение
Источник строк	SELECT DISTINCT Projects.ProjectName FROM Projects ORDER BY Projects.ProjectName
Ограничиться списком	Да

- 4. Откройте модуль формы и введите следующую процедуру обработки события:

```
Private Sub ProjectName_NotInList(NewData As String,  
Response As Integer)  
    'Обработка позиций, отсутствующих  
    'в комбинированном списке ProjectName  
    Dim bytResponse As Byte  
    Dim cnn As ADODB.Connection  
    Dim rst As ADODB.Recordset  
    'On Error GoTo HandleErr  
    bytResponse = MsgBox("Вы хотите добавить "
```



```

    & NewData & "в список?", vbYesNo)
Set cnn = CurrentProject.Connection
Set rst = New ADODB.Recordset
With rst
    .Open "Projects", cnn, adOpenDynamic, _
        adLockOptimistic
    If bytResponse = vbYes Then
        .AddNew "ProjectName", NewData
        .Update
        Response = acDataErrAdded
    ElseIf bytResponse = vbNo Then
        Response = acDataErrContinue
        ProjectName.Undo
        GoTo ExitHere
    End If
End With

ExitHere:
rst.Close
Set rst = Nothing
cnn.Close
Set cnn = Nothing
Exit Sub

HandleErr:
MsgBox "Ошибка " & Err.Number & ": " & _
    Err.Description, vbOKOnly
Resume ExitHere
End Sub

```

5. Сохраните форму и откройте ее в представлении формы. На рис. 17.10 показан только что созданный элемент с открытым списком.

TaskID	TaskName	HourlyRate
1	Управление кадрами	154,50р.
1	Управление контактами	108,15р.
15	Управление проектами	66,95р.
20	Сопровождение	46,35р.

Рис. 17.10. Новый элемент с открытым списком

6. Щелкните на кнопке Новая запись панели навигации формы.
7. Выберите из списка любого клиента.
8. Вместо того, чтобы выбирать название проекта из существующего списка, введите новое название, к примеру "Контроль качества", и нажмите

клавишу <Tab> или <Enter>. При этом возникнет событие отсутствия в списке и откроется диалоговое окно, показанное на рис. 17.11.

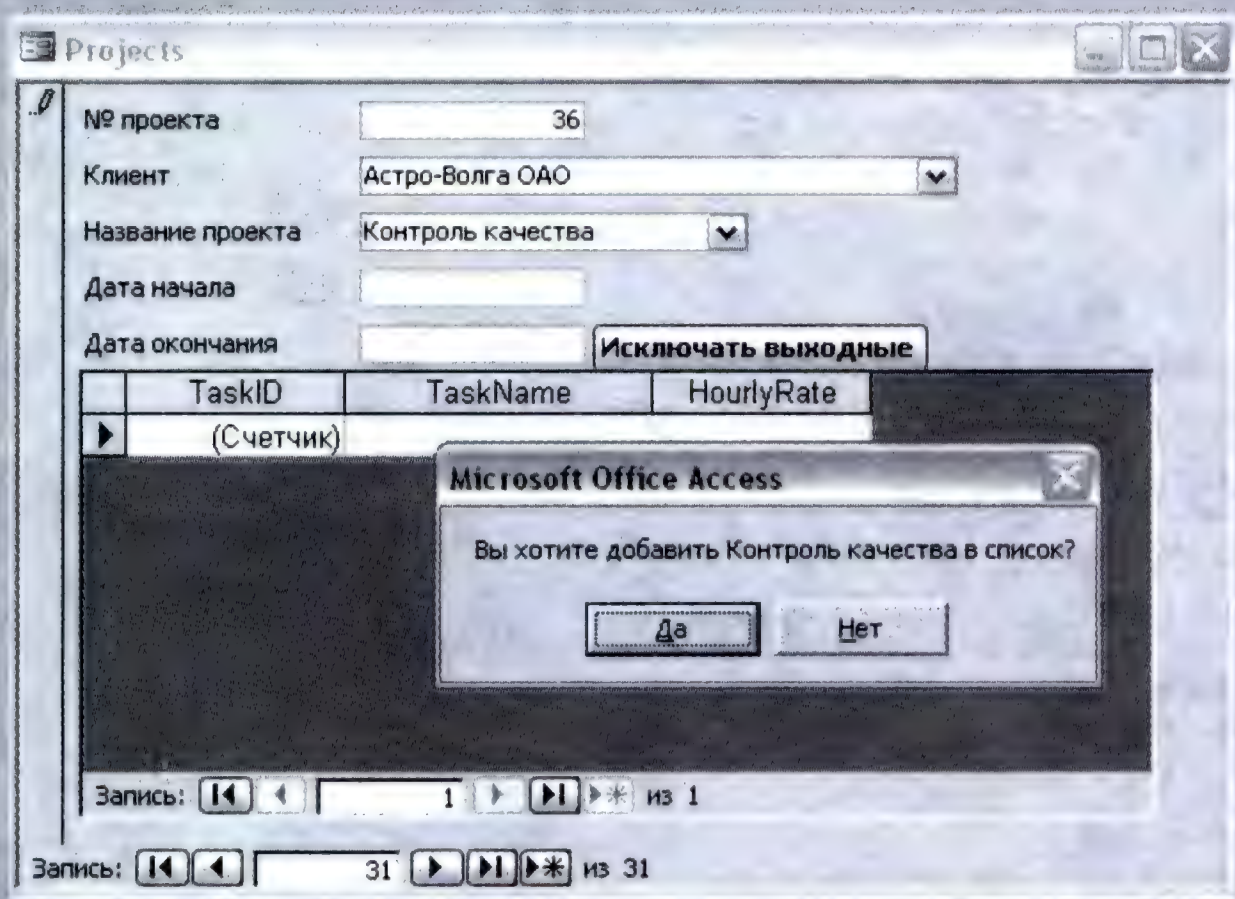


Рис. 17.11. Ввод проекта, отсутствующего в списке, вызывает событие NotInList

- 9. Щелкните на кнопке **Да**.
- 10. Теперь снова откройте этот комбинированный список и вы увидите, что новое название проекта уже добавлено в список (рис. 17.12)

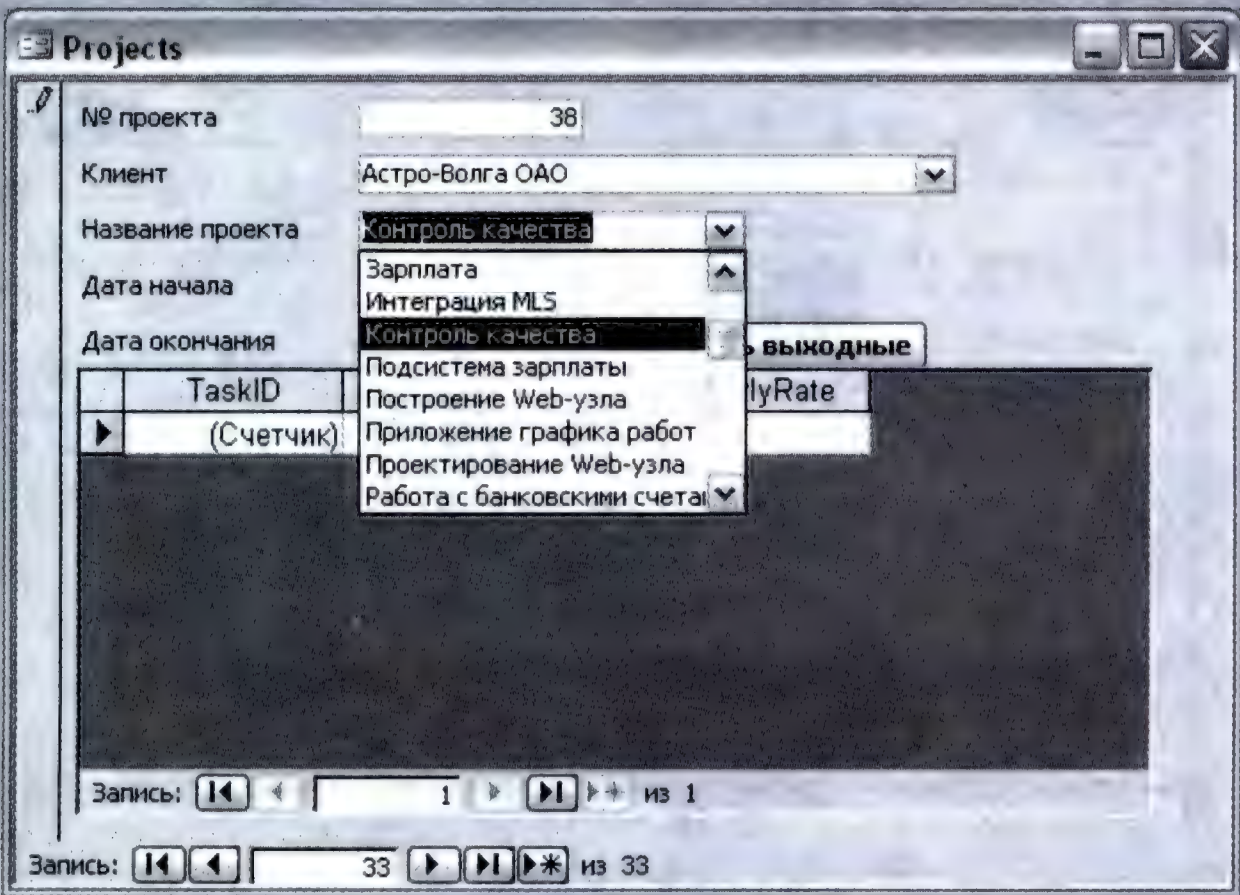


Рис. 17.12. За счет обработки события NotInList мы добавили новый элемент в список

Если в ответ на сообщение, показанное на рис. 17.11, щелкнуть на кнопке **Нет**, процедура обработки удалит эту позицию из текстовой части элемента и не добавит ее в источник данных (таблицу Projects). Следовательно, и в списке элемента эта позиция не появится.

Создание объектов средствами ADOX

18

В предыдущих главах мы учились пользоваться библиотекой ADO для манипулирования данными. Однако с помощью объектов ADO можно выполнить далеко не все операции. Одно из этих ограничений состоит в том, что эта библиотека позволяет работать только с объектами, уже существующими в базе данных. К примеру, средствами ADO нельзя создать новую таблицу или запрос независимо от того, с какими объектами вы имеете дело — Command, Connection или Recordset.

Многие программисты не обращают внимание на наличие этих ограничений, однако порой создание объектов непосредственно из программ оказывается очень полезным. Разве не эффективнее наполнять каждую из создаваемых баз данных некоторым начальным множеством таблиц и отношений? Разве не удобнее для этого использовать всего один программный фрагмент вместо утомительной работы с интерфейсом конструктора таблиц?

Именно с этой целью компания Microsoft выпустила библиотеку расширений ADOX (формально называемую *ADO Extensions for DDL and Security*). Эта дополнительная библиотека позволяет при совместной работе с ADO создавать новые объекты. В этой главы мы опишем некоторые средства и функции библиотеки ADOX.

В ЭТОЙ ГЛАВЕ

Создание таблиц	316
Защита объектов	321

Как и следует из ее названия, библиотека ADOX также позволяет манипулировать свойствами защиты объектов базы данных — например, именами владельцев и списком лиц, имеющих разрешение на работу с объектом.

На рис. 18.1 показан фрагмент объектной модели ADOX, который будет использован в настоящей главе.

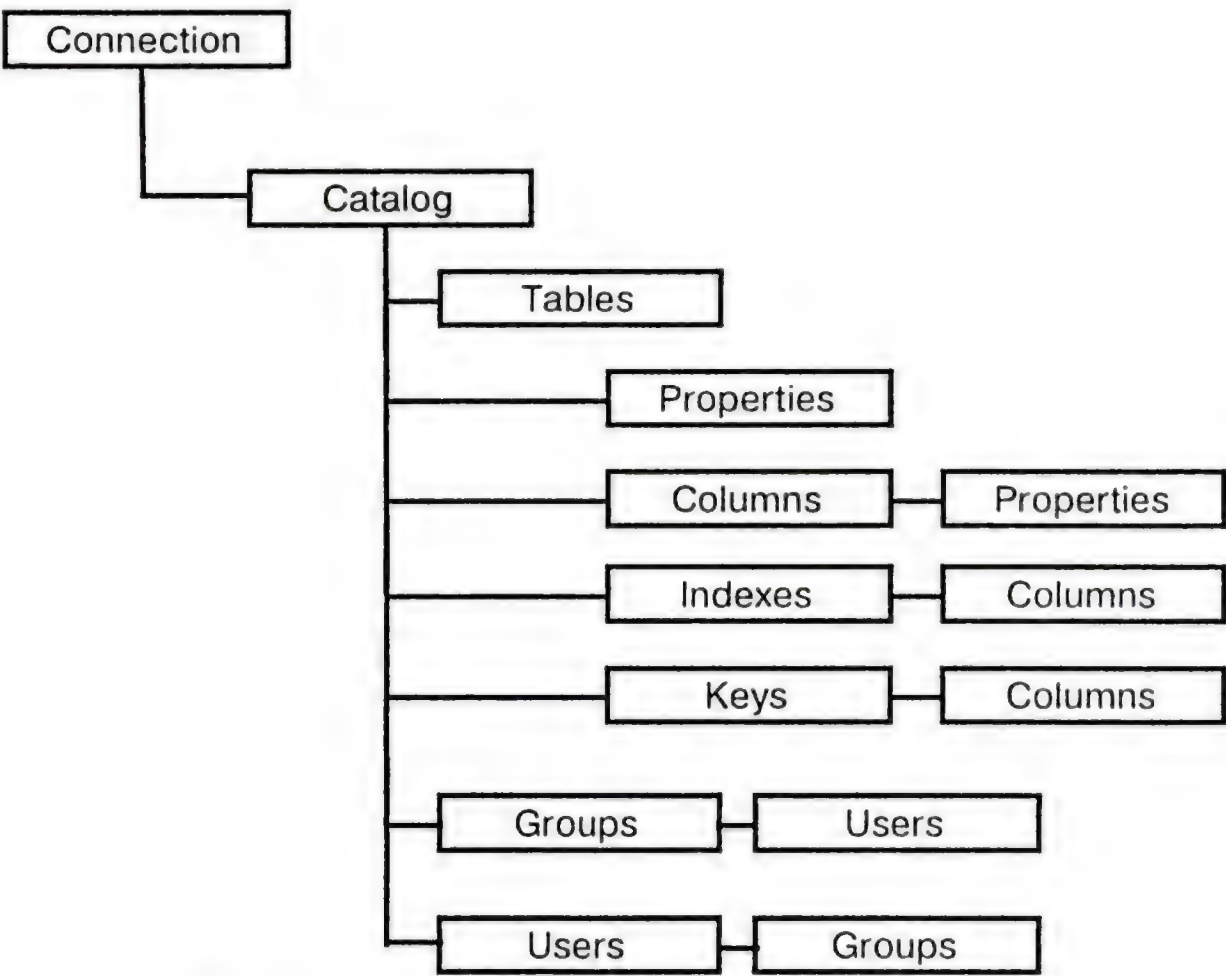


Рис. 18.1. Фрагмент объектной модели ADOX

Примечание
Объект Connection, показанный на рис. 18.1, на самом деле не является частью объектной модели ADOX — это стандартный объект Connection библиотеки ADO. Библиотеки ADO и ADOX совместно используют этот объект для взаимодействия друг с другом.

Создание таблиц

Следующую модель мы будем вновь и вновь использовать при создании различных типов объектов с помощью ADOX:

1. Объявление соответствующего объекта ADOX и его инициализация.
2. Установка значений свойств объекта ADOX.
3. Добавление объекта в соответствующую коллекцию.

Держите эту модель в голове, когда будете создавать новые объекты в тексте программ.

Примечание

Как видите, приведенная схема добавления объектов очень сходна с той, которую мы использовали для добавления данных в объект `Recordset`. Различие состоит лишь в том, что в ADOX работа производится не с данными, а с самой схемой базы данных.

Создание таблицы и ее столбцов

Для создания таблицы потребуется три объекта ADOX:

- Объект `Catalog`, представляющий информацию о конструкции всей базы данных.
- Объект `Table`, представляющий одну таблицу.
- Объект `Column`, представляющий одно поле в таблице.

Чтобы продемонстрировать процесс добавления в базу данных новой таблицы из кода VBA, создадим пару таблиц, которые в дальнейшем можно будет использовать для отслеживания запросов к базе данных. Таблица `Users` (Пользователи) будет состоять из первичного ключа `AutoNumber` и имени пользователя. Таблица `FeatureRequests` будет включать в себя поле авто-нумерации `AutoNumber`, поле вторичного ключа в таблицу `Users` и текстовое поле, хранящее запрос.

Способность создавать такие таблицы в тексте программы очень пригодится программисту, обслуживающему несколько баз данных. Вместо создания вручную одних и тех же таблиц в нескольких базах данных, вы можете импортировать в них один и тот же модуль, после чего просто запустить нужную процедуру. Вот пример процедуры, создающей две вышеописанные таблицы с помощью библиотеки ADOX.

```
Sub CreateTables()  
    ' Создание в базе данных двух новых таблиц  
  
    Dim cat As ADOX.Catalog  
    Dim tbl As ADOX.Table  
    Dim col As ADOX.Column  
  
    ' Установка каталога, указывающего на  
    ' базу данных  
    Set cat = New ADOX.Catalog  
    Set cat.ActiveConnection = CurrentProject.Connection  
  
    ' Создаем таблицу  
    Set tbl = New ADOX.Table  
    tbl.Name = "Users"  
  
    ' Добавляем в таблицу несколько столбцов  
    Set col = New ADOX.Column  
    col.Name = "UserID"  
    col.Type = adInteger      ' длинное целое  
    ' Ассоциируем с поставщиком, чтобы иметь возможность
```



```

' использовать специфичные для поставщика свойства
Set col.ParentCatalog = cat
col.Properties("Autoincrement") = True
col.Properties("Description") = "Уникальный номер пользователя"
tbl.Columns.Append col

Set col = New ADOX.Column
col.Name = "UserName"
col.Type = adVarChar
col.DefinedSize = 50
col.Attributes = 0 ' не пусто
Set col.ParentCatalog = cat
col.Properties("Description") = "Имя пользователя"
tbl.Columns.Append col

' Сохраняем таблицу
cat.Tables.Append tbl

' Создаем вторую таблицу
Set tbl = New ADOX.Table
tbl.Name = "FeatureRequests"

' Добавляем в таблицу столбцы
Set col = New ADOX.Column
col.Name = "FeatureID"
col.Type = adInteger
Set col.ParentCatalog = cat
col.Properties("Autoincrement") = True
col.Properties("Description") = "Уникальный номер запроса"
tbl.Columns.Append col

Set col = New ADOX.Column
col.Name = "UserID"
col.Type = adInteger
Set col.ParentCatalog = cat
col.Properties("Description") = "Пользователь, сделавший запрос"
tbl.Columns.Append col

Set col = New ADOX.Column
col.Name = "FeatureRequest"
col.Type = adVarChar
col.DefinedSize = 255
Set col.ParentCatalog = cat
col.Properties("Description") = "Запрошенное действие"
tbl.Columns.Append col

' Сохраняем таблицу
cat.Tables.Append tbl

End Sub

```

После того как объект `Catalog` проинициализирован, вам нужно установить его свойство `ActiveConnection` (Активное соединение). Эта ссылка указывает на базу данных, в которой будут храниться новосозданные объекты. В данном случае нужную ссылку мы получаем из объекта `CurrentProject` (Текущий проект).

Примечание

Перед запуском этой процедуры следует установить ссылку на библиотеку ADOX. Для этого можно воспользоваться командой меню **Tools⇒References** редактора VBE, после чего установить флажок рядом с названием нужной библиотеки. Учтите, что на компьютере может быть установлено несколько версий библиотеки Microsoft ADO Ext ... for DDL and Security (к примеру, версии 2.6, 2.7 или даже более поздние). В этом случае установите флажок рядом с самой старшей версией библиотеки.

Процесс создания таблицы очень прост. Мы иницилируем новый объект ADOX Table (Таблица), после чего добавляем в него последовательно создаваемые объекты Column (Столбец). Обратите внимание, для того чтобы устанавливать специфичные для Access свойства столбцов, мы в первую очередь устанавливаем свойство ParentCatalog (Родительский каталог). Каждый из столбцов имеет свое имя (свойство Name), тип (свойство Type) и ряд других свойств. (Типы столбцов определяются множеством констант, хранимых в библиотеке ADOX.)

После того как в таблице созданы все столбцы, она добавляется в базу данных путем вставки в соответствующую коллекцию Tables.

Совет

Если раздел Таблицы окна Базы данных открыт во время выполнения описанной выше процедуры, то для того чтобы увидеть новые таблицы, нужно обновить содержимое окна (выполнить команду меню Вид⇒Обновить).

Создание индексов

Простого создания таблицы чаще всего оказывается недостаточно — в большинстве случаев следует создать индексы, которые в дальнейшем сделают работу с таблицами более эффективной. Как правило, в таблицах индексируют все поля, по которым будет впоследствии часто выполняться поиск или сортировка.

В библиотеке ADOX индекс создается с помощью объекта Index, который состоит из коллекции объектов Column, представляющих индексируемые столбцы. Объект Table имеет коллекцию Indexes, хранящую в себе все индексы таблицы. Ниже представлен пример процедуры, добавляющей необходимые индексы в таблицы, созданные в предыдущем разделе.

```
Sub CreateIndexes()  
    ' Создание индексов для новых таблиц  
  
    Dim cat As ADOX.Catalog  
    Dim tbl As ADOX.Table  
    Dim idx As ADOX.Index  
    Dim col As ADOX.Column  
  
    ' Установка каталога, указывающего на  
    ' текущую базу данных
```



```

Set cat = New ADOX.Catalog
Set cat.ActiveConnection = CurrentProject.Connection

' Запрашиваем таблицу
Set tbl = cat.Tables("Users")

' Создаем индекс по полю первичного ключа
' При этом будет создан первичный ключ
Set idx = New ADOX.Index
idx.Name = "PrimaryKey"
idx.PrimaryKey = True
idx.Unique = True
' Задаем столбец для индекса
idx.Columns.Append "UserID"
' и добавляем индекс в таблицу
tbl.Indexes.Append idx

' Создаем второй индекс по полю имени
Set idx = New ADOX.Index
idx.Name = "NameIndex"
idx.Unique = False
idx.Columns.Append "UserName"
tbl.Indexes.Append idx

' Запрашиваем вторую таблицу
Set tbl = cat.Tables("FeatureRequests")

' Создаем ее первичный ключ
Set idx = New ADOX.Index
idx.Name = "PrimaryKey"
idx.PrimaryKey = True
idx.Unique = True
' Задаем столбец для индекса
idx.Columns.Append "FeatureID"
' и добавляем индекс в таблицу
tbl.Indexes.Append idx

```

End Sub

Обратите внимание на некоторые нюансы:

- Запрос уже существующей таблицы ничем не отличается от получения любого другого объекта из соответствующей коллекции.
- Создание объекта `Index` со свойством `PrimaryKey`, установленным в значение `True`, приводит к созданию первичного ключа.
- В примере не потребовалось создавать объекты `Column`, а затем их добавлять в коллекцию `Columns` индекса — все это мы выполнили за одну операцию с помощью метода `Indexes.Append`.

Перед тем как использовать столбцы в качестве индексов, их нужно создать в таблице, в противном случае будет сгенерирована ошибка выполнения.

Создание отношений

Создание отношений между двумя таблицами в ADOX требует создания первичного ключа в одной таблице и вторичного во второй. Оба эти ключа

могут быть представлены объектами Key. Как уже говорилось ранее, при создании индекса первичного ключа автоматически создается сам первичный ключ. Для построения отношения остается создать вторичный ключ.

```
Sub CreateRelation()
    ' Создание вторичного ключа
    ' между таблицами Users и FeatureRequests

    Dim cat As ADOX.Catalog
    Dim tbl As ADOX.Table
    Dim ky As ADOX.Key
    Dim col As ADOX.Column

    ' Установка указателя каталога на
    ' текущую базу данных
    Set cat = New ADOX.Catalog
    Set cat.ActiveConnection = CurrentProject.Connection

    ' Запрашиваем таблицу вторичного ключа
    Set tbl = cat.Tables("FeatureRequests")

    ' Создаем ключ и добавляем его в таблицу
    Set ky = New ADOX.Key
    ky.Name = "UserKey"
    ky.Type = adKeyForeign
    ky.RelatedTable = "Users"
    Set col = New ADOX.Column
    col.Name = "UserID"
    col.RelatedColumn = "UserID"
    ky.Columns.Append col
    tbl.Keys.Append ky

End Sub
```

Объект Key, представляющий вторичный ключ, имеет свойства, отражающие обе стороны отношения:

- Свойство Column.Name является именем столбца таблицы вторичного ключа.
- Свойство Column.RelatedColumn является именем столбца таблицы первичного ключа.
- Свойство Key.RelatedTable является именем таблицы первичного ключа.
- Сам ключ добавляется в коллекцию Keys таблицы вторичного ключа.

Защита объектов

Создание таблиц, индексов и отношений являются видами деятельности, которые влияют на схему базы данных, т.е. на компоновку ее объектов. Все это относится к одной из функций библиотеки ADOX, упомянутой в ее названии — “Data Definition” (Определение данных). В то же время библиотека ADOX содержит и функции манипулирования свойствами безопасности базы данных. Все эти функции в точности соответствуют методике управления

безопасностью в Access. Средства безопасности библиотеки ADOX позволяют выполнять следующие операции:

- создавать группы;
- создавать пользователей;
- изменять собственника объекта;
- устанавливать права доступа к объекту.

В этом разделе будут показаны основные программные конструкции, позволяющие выполнять эти операции.

Создание новой группы

Приложение Access позволяет устанавливать такую защиту объектов, чтобы они были доступны только конкретным пользователям или группам пользователей. Начнем рассмотрение данного вопроса именно с групп, так как с их помощью в большинстве случаев управлять системой безопасности гораздо удобнее. Если у базы данных имеется множество пользователей, вы можете установить права доступа для групп, после чего назначать пользователей соответствующим группам. Для того чтобы создать новый объект группы безопасности Group с помощью библиотеки ADOX, ему нужно присвоить имя, после чего добавить в коллекцию Groups каталога, как в следующем примере:

```
Sub CreateGroup()
    ' Создание новой группы безопасности

    Dim cat As ADOX.Catalog

    ' Установка указателя каталога на текущую
    ' базу данных
    Set cat = New ADOX.Catalog
    Set cat.ActiveConnection = CurrentProject.Connection

    ' Добавляем группу с именем Management
    cat.Groups.Append "Management"
    ' Добавляем в эту группу пользователя Admin
    cat.Groups("Management").Users.Append "Admin"

End Sub
```

Предупреждение

Примеры, приведенные в данном разделе, внесут изменения в рабочую группу Access, созданную в приложении по умолчанию. В модуле Chapter18 базы данных примеров TimeTrack содержится специальная процедура Cleanup, позволяющая отменить все проводимые процедурами этого раздела изменения.

После создания группы в нее можно добавлять пользователей. Для этого используется метод Add коллекции Users группы. Добавляемые пользователи уже должны существовать в коллекции Users каталога. Подробно процесс добавления пользователей будет рассмотрен в следующем подразделе.

Создание новых пользователей

Приведенный ниже пример добавления нового пользователя очень сходен с примером добавления новой группы.

```
Sub CreateUser()
    ' Создание нового пользователя в системе безопасности

    Dim cat As ADOX.Catalog

    ' Установка указателя каталога на текущую
    ' базу данных
    Set cat = New ADOX.Catalog
    Set cat.ActiveConnection = CurrentProject.Connection

    ' добавляем пользователя с именем Mike
    cat.Users.Append "Mike", "OriginalPW"
    ' и относим его к группе Management
    cat.Users("Mike").Groups.Append "Management"

    ' Изменяем пароль нового пользователя
    cat.Users("Mike").ChangePassword "OriginalPW", "Fish"

End Sub
```

После того как пользователь создан, помещаем его в группу. Для этого мы добавляем группу в коллекцию Groups объекта данного пользователя (разумеется, сама группа должна уже существовать). В качестве альтернативы вы можете добавить пользователя в коллекцию Users объекта группы. В обоих случаях результат операции будет одним и тем же.

При создании нового пользователя ему нужно присвоить пароль. Если вы передумаете насчет самого пароля, то можете воспользоваться методом ChangePassword объекта User и этот пароль изменить (для использования этого метода нужно знать старый пароль).

Предупреждение

В реальных приложениях пароли обычно не хранят в тексте программы, иначе любой пользователь, которому попадет в руки текст программы или файл MDB, сможет прочесть этот пароль. Более безопасным способом является запрос желаемого пароля у самого пользователя в ходе выполнения программы и сохранение его в базе данных в зашифрованном виде.

Изменение владельца объекта

Любой объект в Access изначально принадлежит тому пользователю, который его создал. При изменении владельца объекта коллекции не используются (ни Groups, ни Users). Извлечение информации о владельце объекта, а также назначение прав владельца другому пользователю осуществляется с помощью методов объекта Catalog. Приведем пример.

```
Sub ChangeOwner()
    ' Изменение владельца объекта
```



```

Dim cat As ADOX.Catalog
Dim strOwner As String

' Установка указателя каталога на текущую
' базу данных
Set cat = New ADOX.Catalog
Set cat.ActiveConnection = CurrentProject.Connection

' Показываем текущего владельца таблицы Clients
strOwner = cat.GetObjectOwner("Clients", adPermObjTable)
Debug.Print "Таблица клиентов принадлежит пользователю" &
strOwner

' Изменяем владельца
cat.SetObjectOwner "Clients", adPermObjTable, "Mike"
Debug.Print "А теперь таблица Clients принадлежит пользователю
Mike"

End Sub

```

Методы `GetObjectOwner` и `SetObjectOwner` принимают в качестве одного из параметров тип объекта, что указывает им на то, с каким типом объектов базы данных придется работать. При работе в Access этот параметр может принимать значения следующих констант:

- `adPermObjProcedure` для запроса с параметрами;
- `adPermObjTable` для таблицы;
- `adPermObjView` для запроса без параметров;
- `adPermObjColumn` для столбца;
- `adPermObjDatabase` для базы данных;
- `adPernObjProviderSpecific` для всех остальных объектов.

Если методам передается константа `adPernObjProviderSpecific`, им следует предоставить дополнительный параметр:

```

cat.GetObjectOwner имя_объекта, тип_объекта, идентификатор
cat.GetObjectOwner имя_объекта, тип_объекта, пользователь, _
идентификатор

```

В данном случае параметром *идентификатор* выступает глобальный уникальный идентификатор *GUID*. Вы можете использовать следующие идентификаторы:

- {C49C842E-9DCB-11D1-9F0A-00C04FC2C2E0} для форм;
- {C49C8430-9DCB-11D1-9F0A-00C04FC2C2E0} для отчетов;
- {C49C842F-9DCB-11D1-9F0A-00C04FC2C2E0} для макросов;
- {C49C8432-9DCB-11D1-9F0A-00C04FC2C2E0} для модулей.

Двухэтапный процесс (выбор константы `adPernObjProviderSpecific`, а затем установка одного из идентификаторов GUID) необходим по той при-

чине, что библиотека ADOX работает также и с базами данных, отличными от Access (например, с SQL Server и Oracle). В этих базах данных не существует таких объектов, как формы и отчеты, поэтому не было никакой необходимости включать их в основной набор констант библиотеки ADOX.

Установка прав доступа к объектам

Для установки прав доступа к объекту используется метод `SetPermissions` объекта пользователя или группы. Приведем пример:

```
Sub SetPermissions()
    ' Установка прав доступа

    Dim cat As ADOX.Catalog

    ' Установка указателя каталога на текущую
    ' базу данных
    Set cat = New ADOX.Catalog
    Set cat.ActiveConnection = CurrentProject.Connection

    ' Установка для пользователя Mike прав доступа к
    ' таблице Clients
    cat.Users("Mike").SetPermissions "Clients", adPermObjTable, _
        adAccessSet, adRightFull

End Sub
```

Метод `SetPermissions` имеет четыре обязательных и два необязательных аргумента:

- *имя объекта*, с которым проводится манипуляция;
- *тип объекта*;
- *действие* — принимает значение одной из следующих констант: `adAccessDeny` для снятия определенных прав, `adAccessGrant` для добавления заданных прав к уже существующим, `adAccessRevoke` для снятия абсолютно всех прав или `adAccessSet` для установки точного набора прав;
- *права доступа* — принимает значение одной из констант, определенных в табл. 18.1;
- необязательный аргумент *наследование* определяет порядок наследования дочерними объектами прав, устанавливаемых или снимаемых для данного объекта, и принимает значение одной из констант (скорее всего, вам этот параметр не понадобится);
- необязательный параметр *идентификатор типа объекта* используется, когда аргумент *типа объекта* установлен в константу `adPermObjProviderSpecific` (подробно идентификаторы объектов рассматривались в предыдущем подразделе).

Таблица 18.1. Константы прав доступа в библиотеке ADOX

Константа	Значение
adRightCreate	Разрешение создания объектов определенного типа
adRightDelete	Разрешение удаления данных
adRightDrop	Разрешение удаления объектов определенного типа
adRightExclusive	Разрешение монопольного блокирования объекта
adRightExecute	Разрешение выполнения
adRightFull	Все возможные права доступа к объекту
adRightInsert	Разрешение вставки данных
adRightMaximumAllowed	Все возможные права доступа, включая специфич- ные для поставщика
adRightNone	Полное отсутствие прав доступа к объекту
adRightRead	Разрешение на чтение данных
adRightReadDesign	Разрешение на извлечение информации о схеме
adRightReadPermissions	Разрешение на извлечение прав доступа
adRightReference	Разрешение на ссылку на объект
adRightUpdate	Разрешение на редактирование существующих данных
adRightWithGrant	Разрешение на установку прав доступа
adRightWriteDesign	Разрешение на изменение конструкции объекта
adRightWriteOwner	Разрешение на изменение владельца объекта
adRightWritePermissions	Разрешение на изменение установленных прав доступа

ПРАКТИКУМ

Создание словаря данных

Для демонстрации практического использования библиотеки ADOX в предлагае-
мом практикуме мы создадим словарь данных. *Словарем данных* называют спи-
сок всех полей таблиц базы данных наряду с некоторой информацией об этих по-
лях. Чаще всего такой список используют для проверки целостности базы данных.
К примеру, причиной ошибок в работе программ может быть то, что в одних таб-
лицах идентификатор поставщика определен как числовое значение, в других
же — как строка.

В нашей реализации словаря данных мы отобразим имя столбца, имя таблицы
и соответствующий тип данных. Словарь будет организован в виде одной таблицы

и одной формы. Его впоследствии можно будет использовать в любой базе данных путем импорта таблицы и формы и последующего открытия формы. Вот как создать словарь данных с нуля:

1. Создайте новую таблицу, содержащую три поля: TableName (текстовое поле, 255 символов), ColumnName (также текстовое поле, 255 символов) и Type (текстовое поле, 50 символов). Присвойте таблице имя DataDictionary.
2. Создайте новую форму и присвойте ей имя Data Dictionary. Установите свойства Область перехода и Кнопки перехода в значение Нет, а свойство Подпись — в Словарь данных.
3. Добавьте в форму элемент простого списка и присвойте ему имя lstColumns. Установите свойство Число столбцов в значение 3, а свойство Источник строк — в выражение `SELECT DataDictionary.ColumnName, DataDictionary.TableName, DataDictionary.Type FROM DataDictionary ORDER BY DataDictionary.ColumnName, DataDictionary.TableName`.
4. В модуль формы добавьте следующие процедуры:

```
Option Compare Database
```

```
Option Explicit
```

```
Private Sub Form_Open(Cancel As Integer)
```

```
    ' Перестройка словаря при открытии формы
```

```
    Dim cnn As ADODB.Connection
```

```
    Dim cmdDelete As ADODB.Command
```

```
    Dim rstDictionary As ADODB.Recordset
```

```
    Dim cat As ADOX.Catalog
```

```
    Dim tbl As ADOX.Table
```

```
    Dim col As ADOX.Column
```

```
    ' Очистка существующего словаря
```

```
    Set cnn = CurrentProject.Connection
```

```
    Set cmdDelete = New ADODB.Command
```

```
    Set cmdDelete.ActiveConnection = cnn
```

```
    cmdDelete.CommandText = "DELETE * FROM DataDictionary"
```

```
    cmdDelete.Execute
```

```
    ' Заполнение набора данных для нового словаря
```

```
    Set rstDictionary = New ADODB.Recordset
```

```
    rstDictionary.Open "SELECT * FROM DataDictionary", _  
        cnn, adOpenStatic, adLockOptimistic
```

```
    ' Установка указателя каталога в
```

```
    ' текущую базу данных
```

```
    Set cat = New ADOX.Catalog
```

```
    Set cat.ActiveConnection = CurrentProject.Connection
```

```
    ' Цикл по всем таблицам и полям
```

```
    For Each tbl In cat.Tables
```

```
        ' Блокировка системных таблиц
```

```
        If Left(tbl.Name, 4) <> "MSys" And _
```

```
            Left(tbl.Name, 4) <> "USys" And _
```

```
            Left(tbl.Name, 4) <> "~TMP" Then
```

```
            For Each col In tbl.Columns
```



```

        rstDictionary.AddNew
        rstDictionary.Fields("TableName").Value = tbl.Name
        rstDictionary.Fields("ColumnName").Value = col.Name
        rstDictionary.Fields("Type").Value = _
            TranslateType(col.Type)
        rstDictionary.Update
    Next col
End If
Next tbl

rstDictionary.Close
End Sub

Public Function TranslateType(intType As Integer) As String
    ' Получение типа данных Access, соответствующего
    ' типу данных ADOX
    Select Case intType
        Case adUnsignedTinyInt
            TranslateType = "Byte"
        Case adCurrency
            TranslateType = "Currency"
        Case adDate
            TranslateType = "Date/Time"
        Case adNumeric
            TranslateType = "Decimal"
        Case adDouble
            TranslateType = "Double"
        Case adLongVarChar
            TranslateType = "Memo"
        Case adSmallInt
            TranslateType = "Integer"
        Case adInteger
            TranslateType = "Long Integer"
        Case adLongVarBinary
            TranslateType = "OLE Object"
        Case adGUID
            TranslateType = "Replication ID"
        Case adSingle
            TranslateType = "Single"
        Case adVarChar
            TranslateType = "Text"
        Case adBoolean
            TranslateType = "Yes/No"
    End Select
End Function

Private Sub Form_Load()
    ' После переформирования набора данных
    ' обновляем элемент списка
    lbColumns.Requery
End Sub

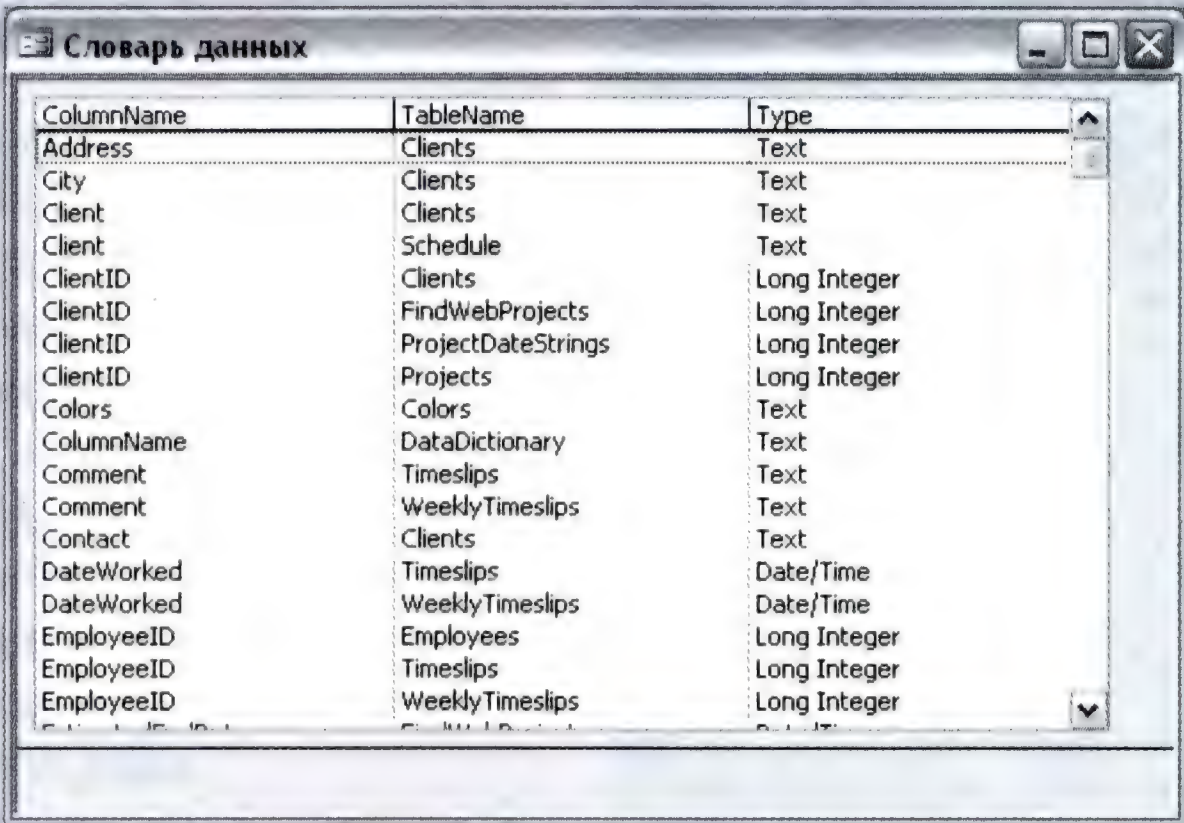
```

На первый взгляд приведенная программа имеет сложный вид, однако она сводит воедино рассмотренные нами технологии работы с библиотеками ADO и ADOX. При открытии формы выполняется метод Execute объекта Command для очистки таблицы DataDictionary, что гарантирует факт обновления таблицы при каж-

дом открытия формы. После этого с помощью библиотеки ADO на базе обновленной таблицы мы открываем набор данных.

Программа переключается на библиотеку ADOX для прохождения в цикле по всем таблицам в базе данных и по всем столбцам таблиц. В теле цикла установлена проверка на временные объекты и системные таблицы во избежание их перечисления в словаре данных. Все эти элементы имеют в своих названиях характерный набор префиксов. При прохождении каждого из столбцов допустимых таблиц формируется новая строка в наборе данных.

Функция TranslateType переводит константы ADOX в более привычные типы данных Access. После того как весь набор данных был заполнен, элемент списка на форме обновляется (рис. 18.2).



ColumnName	TableName	Type
Address	Clients	Text
City	Clients	Text
Client	Clients	Text
Client	Schedule	Text
ClientID	Clients	Long Integer
ClientID	FindWebProjects	Long Integer
ClientID	ProjectDateStrings	Long Integer
ClientID	Projects	Long Integer
Colors	Colors	Text
ColumnName	DataDictionary	Text
Comment	Timeslips	Text
Comment	WeeklyTimeslips	Text
Contact	Clients	Text
DateWorked	Timeslips	Date/Time
DateWorked	WeeklyTimeslips	Date/Time
EmployeeID	Employees	Long Integer
EmployeeID	Timeslips	Long Integer
EmployeeID	WeeklyTimeslips	Long Integer

Рис. 18.2. Словарь данных в действии

Совет

В предложенной методике следует обратить внимание на тот факт, что не существует встроенных функций или свойств, которые сортируют свободный список в форме. Простейшим способом получения отсортированного списка является сохранение данных в промежуточной таблице и последующее извлечение тех же данных с применением предложения ORDER BY.

Расширенные операции с данными

Программирование конкуренции

Приложение Access изначально создавалось для работы с базами данных в многопользовательском режиме. Это значит, что несколько пользователей одновременно могут открыть одну и ту же базу данных, и это ни к чему плохому не приведет. С точки зрения интерфейса пользователя Access этот тезис является истинным; что касается программ автоматизации Access, это также не противоречит истине.

Ключевым понятием в многопользовательской работе с базами данных является *конкуренция*: что произойдет, если два пользователя попытаются отредактировать в базе данных одну и ту же запись одновременно? В этом разделе кратко описаны способы работы с конкуренцией в Access, а также показано, как подходить к этому вопросу в программах, использующих библиотеку ADO.

Понятие конкуренции

Существует две основные методики работы приложений (в частности, Access) с вопросами конкуренции. Обычно их называют *оптимистической* и *пессимистической блокировкой записей*. Блокировка записей представляет со-

В ЭТОЙ ГЛАВЕ

Программирование конкуренции	331
Извлечение набора данных пользователей	337
Использование других наборов данных схем	339

бой процесс резервирования записи для монопольного использования одним пользователем базы данных. Любая блокировка всегда ассоциируется с конкретной записью в таблице и конкретным пользователем. Когда запись заблокирована, ни один другой пользователь не сможет ее изменить.

Различие между оптимистической и пессимистической блокировками состоит в том, в какой момент эта блокировка устанавливается и снимается. В первом случае блокировка устанавливается только на период сохранения изменений; во втором же — на весь период редактирования записи. В любом случае, Access снимает блокировку после сохранения изменений.

Примечание

В старых версиях приложения Access для блокировки записей использовалась *блокировка страниц*. При этом блокировка одной записи приводила к блокировке и соседних, расположенных на одной с ней странице памяти. Начиная с версии Access 2000 по умолчанию применяется блокировка записей, а не страниц. При желании вы можете изменить это поведение. Для этого выберите пункт меню Сервис⇒Параметры (Tools⇒Options) и в закладке Другие (Advanced) поставьте переключатель Блокировка по умолчанию в любую из позиций, отличных от Изменяемой записи (хотя на самом деле лучше все оставить как есть).

Оптимистическая блокировка позволяет одновременно работать над одной и той же записью нескольким пользователям. Вот пример возможной последовательности действий при оптимистической блокировке:

1. Петр начинает редактирование записи в таблице заказчиков записи Совкомбанк.
2. Петр изменяет номер телефона этой компании.
3. Яна также начинает редактировать запись Совкомбанка. Она может это сделать, поскольку в момент начала редактирования записи Петр еще не установил блокировку.
4. Теперь Яна изменяет номер телефона компании.
5. Петр сохраняет изменения. На время записи изменений на диск запись блокируется. После записи номер телефона клиента становится таким, каким его ввел Петр.
6. Яна пытается сохранить изменения. Она получает сообщение об ошибке, информирующее, что другой пользователь уже изменил запись. Теперь у Яны есть выбор: отменить свои изменения или переписать поверх обновления Петра.

С другой стороны, если в таблице использована пессимистическая блокировка записей, та же последовательность действий примет другой вид:

1. Петр начинает редактирование записи заказчика Совкомбанк.
2. Петр изменяет номер телефона клиента.

3. Яна пытается начать редактирование этой же записи, но поскольку запись заблокирована Петром, ей это не удастся.
4. После того как Петр сохранил изменения, Яна уже может отредактировать запись по своему усмотрению.

При пессимистической блокировке записей пользователи базы данных могут блокировать работу друг друга. В приведенном выше примере Яна не могла внести свои изменения до тех пор, пока это не сделает Петр, даже если изменять она собиралась совсем другое поле записи. Оптимистическая блокировка имеет то преимущество, что пользователи не стоят друг у друга на дороге.

В то же время у оптимистической блокировки существует и характерный недостаток. Несмотря на то, что явной блокировки работы другого пользователя не происходит, те, кто пытается сохранить запись первыми, имеют все шансы потерять результат своей работы. К примеру, если два пользователя работают над одной и той же записью, побеждает из них тот, кто сохранит изменения последним.

Совет

В большинстве случаев все же следует выбирать оптимистическую блокировку. Это связано с тем, что в крупных базах данных вероятность одновременной работы двух разных пользователей с одними и теми же данными исключительно мала.

Оптимистическая блокировка в ADO

Оптимистическая блокировка используется по умолчанию в объектах наборов данных ADO. Вот пример программы, использующей оптимистическую блокировку:

```
Public Sub TestOptLock()
    ' Тестирование оптимистической блокировки
    Dim rstOpt As ADODB.Recordset
    Set rstOpt = New ADODB.Recordset

    ' Открытие набора записей с оптимистической блокировкой
    rstOpt.Open "Clients", CurrentProject.Connection, _
        adOpenKeyset, adLockOptimistic, adCmdTable

    ' Останавливаем выполнение для тестирования
    Stop

    ' Изменяем данные и сохраняем запись
    rstOpt.Fields("State").Value = "Россия"
    ' Блокировка устанавливается только на время
    ' выполнения следующей инструкции
    rstOpt.Update

    rstOpt.Close
End Sub
```


Обратите внимание на инструкцию `Stop` в середине процедуры. Данный оператор указывает интерпретатору VBA, что в этой точке нужно временно приостановить выполнение процедуры. Чем-то это похоже на установку в тексте программы контрольной точки. Это поможет нам провести эксперимент с оптимистической блокировкой без необходимости запуска второй копии приложения Access. Для наших исследований придется выполнить описанные далее действия.

1. Откройте модуль `Chapter19` в базе данных примеров `TimeTrack.mdb` и запустите на выполнение процедуру `TestOptLock` в окне `Immediate`.
2. Теперь вернитесь в интерфейс пользователя Access и откройте таблицу `Clients` из окна `База данных`.
3. Измените регион первой записи на любой другой (к примеру, на `Чукотка`), но при этом не сохраняйте изменений и не переходите к другой записи.
4. Вернитесь к модулю и нажмите клавишу `<F5>`. Интерпретатор завершит выполнение процедуры и сохранит изменения в записи (в процедуре мы устанавливаем значение региона в `Россия`). В программе все изменения будут сохранены без проблем, поскольку интерфейс пользователя Access использует оптимистическую блокировку записей по умолчанию.
5. Вернитесь в интерфейс пользователя Access и попытайтесь сохранить запись. Результатом станет сообщение об ошибке, показанное на рис. 19.1.

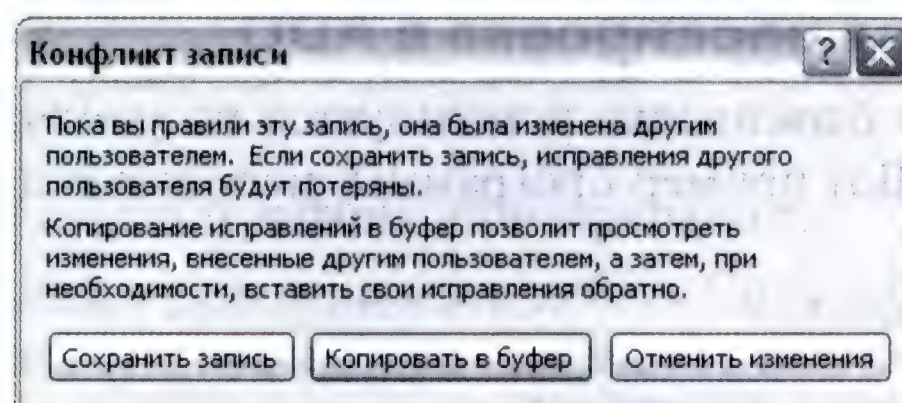


Рис. 19.1. Оптимистическая блокировка вызвала появление сообщения о конфликте

В этом сообщении вам предлагается три варианта действий:

- Сохранение собственных изменений, перекрывающих изменения другого пользователя.
- Копирование собственных изменений в буфер обмена. При этом происходит автоматическое обновление данных в используемом вами представлении с отображением изменений, выполненных другим пользователем. Если вам не нравятся изменения, проведенные другим пользователем, вы можете вставить поверх них данные из буфера обмена.
- Отказ от собственных изменений.

Теперь вы знаете, как выглядит конфликт со стороны интерфейса пользователя Access. Давайте посмотрим на него с обратной стороны. Для этого введите в стандартный модуль следующую процедуру:


```

Public Sub TestOptLock2()
    ' Тестирование оптимистической блокировки
    Dim rstOpt As ADODB.Recordset
    Set rstOpt = New ADODB.Recordset

    ' Открытие набора записей с оптимистической блокировкой
    rstOpt.Open "Clients", CurrentProject.Connection, _
        adOpenKeyset, adLockOptimistic, adCmdTable

    ' Изменяем данные и сохраняем их
    rstOpt.Fields("State").Value = "Чукотка"
    ' для эксперимента останавливаем программу
    Stop

    ' Блокировка только на время работы
    ' следующей инструкции
    rstOpt.Update

    rstOpt.Close
End Sub

```

Различие между процедурой TestOptLock2 и предыдущей состоит в том, что теперь оператор останова выполнения переместился в место после выполнения изменений в записи. Проследим эффект этого изменения.

1. Восстановите первую запись таблицы `Clients` в ее исходном состоянии.
2. Запустите процедуру TestOptLock2 из окна Immediate. Она остановится в месте инструкции `Stop`.
3. Теперь вернитесь в интерфейс пользователя Access и откройте таблицу `Clients` из окна База данных.
4. Измените регион в первой записи таблицы на любой другой, к примеру на Татарстан. Перейдите к другой записи, чтобы сохранить изменения. Интерфейс пользователя заблокирует запись только на период сохранения изменений.
5. Вернитесь в модуль и нажмите клавишу <F5> для продолжения выполнения программы. Однако вместо завершения выполнения процедуры вы получите сообщение об ошибке, показанное на рис. 19.2.

Естественно, если в процедуру добавить код обработки ошибок, можно предотвратить стандартную обработку ошибки конфликта записи. В данном же случае вы сами должны решить, как обрабатывать ошибку и сообщать об этом пользователю.

Если внимательно исследовать две только что использованные процедуры, можно увидеть, как именно приложение Access обнаруживает конфликт. Когда вы редактируете запись в первый раз, Access сохраняет копию данных в исходном виде. Когда вы пытаетесь сохранить запись, перед выполнением сохранения Access сравнивает данные на жестком диске с их сохраненной копией. Если обнаруживается несоответствие (т.е. кто-то уже успел изменить первоисточник), генерируется ошибка конфликта записи.

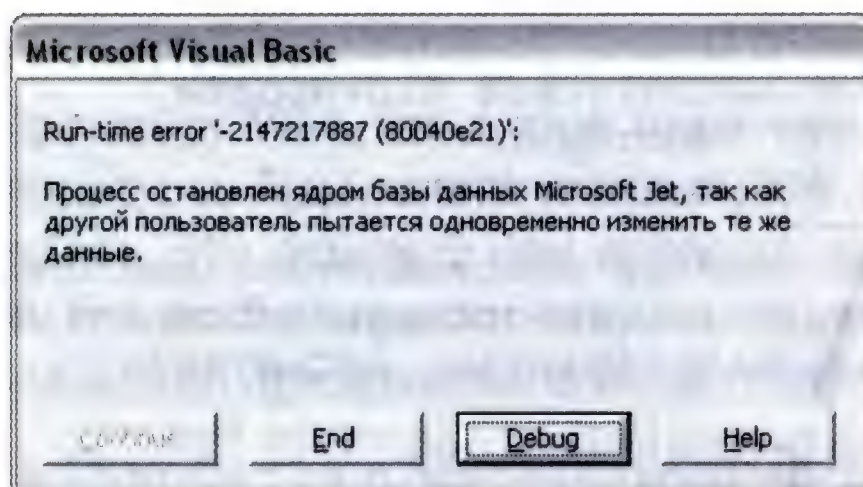


Рис. 19.2. Оптимистическая блокировка привела к ошибке выполнения программы

Пессимистическая блокировка в ADO

Программа для тестирования пессимистической блокировки имеет более сложный вид:

```
Public Sub TestPessLock()
    ' Тест пессимистической блокировки
    Dim cnn As ADODB.Connection
    Dim rstOpt As ADODB.Recordset

    ' Подключаемся к текущему проекту
    Set cnn = New ADODB.Connection
    cnn.Open CurrentProject.BaseConnectionString

    Set rstOpt = New ADODB.Recordset

    ' Открываем набор записей с пессимистической блокировкой
    rstOpt.Open "Clients", cnn, _
        adOpenKeyset, adLockPessimistic, adCmdTable

    ' Изменяем данные и сохраняем их
    ' Блокировка начинается в момент выполнения следующей инструкции
    rstOpt.Fields("State").Value = "Чукотка"
    ' Останавливаем выполнение для проведения эксперимента
    Stop

    rstOpt.Update

    rstOpt.Close
End Sub
```

Предупреждение

В этой процедуре открывается новое подключение к текущему проекту, а не дается ссылка на объект `CurrentProject.Connection`. Это объясняется тем, что невозможно открыть набор данных `Recordset` на текущем подключении с пессимистической блокировкой записей. Если вы все же попытаетесь это сделать, никакой ошибки не произойдет — просто Access негласно изменит блокировку на оптимистическую.

Чтобы понять, как данная процедура взаимодействует с другими попытками изменения данных, выполним следующее упражнение.

1. Восстановите первую запись таблицы `Clients` в ее исходном состоянии.
2. Запустите процедуру `TestPessimLock` из окна `Immediate`. Ее выполнение остановится на инструкции `Stop`. В этой точке процедура уже заблокировала первую запись таблицы (при пессимистической блокировке запись блокируется во время первой попытки ее редактирования).
3. Теперь вернитесь в интерфейс пользователя `Access` и откройте таблицу `Clients` из окна `База данных`.
4. Измените регион в первой записи на любой другой. При попытке перехода к другой записи или сохранении изменений откроется сообщение об ошибке, показанное на рис. 19.3. Щелкните на кнопке `ОК` и закройте окно сообщения.
5. Вернитесь в модуль и нажмите клавишу `<F5>`, чтобы продолжить выполнение процедуры. Теперь `Access` благополучно сохранит изменения, проведенные процедурой.
6. Вернитесь в интерфейс пользователя `Access` и снова попытайтесь сохранить запись. Откроется диалоговое окно конфликта записи (см. рис. 19.1), так как только что были сохранены изменения, проведенные программой.

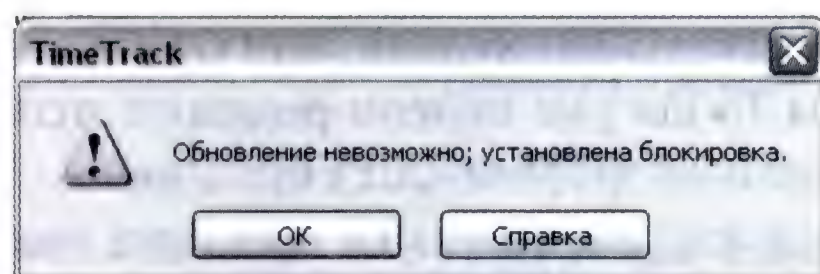


Рис. 19.3. Сообщение пессимистической блокировки

Приложение `Access` позволило редактировать запись последовательно двум действующим лицам. Как видите, на шаге 4 сохранить изменения не удалось. Находясь в интерфейсе `Access`, вы работаете с копией данных до тех пор, пока не попытаетесь ее сохранить. Именно в этот момент `Access` проверяет наличие конфликта записи.

➔ Существует еще две константы блокировки, кроме упомянутых в этом разделе. Краткое описание этих констант вы можете найти в разделе “Создание и открытие набора записей” главы 16.

Извлечение пользовательского набора данных

До сих пор большая часть объектов наборов данных, с которыми мы работали, имели своим источником базу данных. Однако существует еще один тип объектов `Recordset` — *наборы данных схем*. Эти объекты содержат информацию о схеме (конструкции) базы данных и некоторую другую информацию, специфичную для механизма работы с базой данных.

Для открытия набора данных схем используется метод `OpenSchema` объекта `Connection`. Этот метод имеет три аргумента:

- *тип запроса* — константа, указывающая на тип извлекаемой информации;
- *критерий* — необязательный фильтр, ограничивающий объем информации, возвращаемой ADO;
- *идентификатор схемы* — если в качестве типа запроса использована константа `adSchemaProviderSpecific`, этот аргумент является обязательным и указывает на конкретный набор данных схемы, который следует вернуть.

Дальнейшее изложение материала немного прояснит использование аргумента идентификатора схемы. Создатели библиотеки ADO определили множество стандартных схематических наборов данных, которые, по их мнению, способны удовлетворить пользователей большинства типов баз данных (к примеру, набор данных со списком всех таблиц). В то же время, создатели ADO не могли учесть все схематические наборы, используемые разными базами данных. По этой причине была введена константа `adSchemaProviderSpecific` — нечто вроде символа макроподстановки. Если вызывается этот тип схемы, реальный, специфический для конкретного поставщика тип задается аргументом идентификатора схемы. Это значение, как правило, является *глобальным уникальным идентификатором*, или *GUID*.

В практикуме главы 16 вы уже видели реальное использование идентификатора схемы при создании формы `CurrentUsers`. Эта форма использует набор данных схемы для извлечения имен всех пользователей, у которых в настоящий момент открыта база данных (см. модуль `Chapter16` базы данных примеров):

```
Public Const JET_SCHEMA_USERROSTER =
    "{947bb102-5d43-11d1-bdbf-00c04fb92675}"

Public Function ReturnUsers() As String
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = New ADODB.Connection
    On Error GoTo HandleErr
    'Подключаемся к базе данных
    Set cnn = CurrentProject.Connection
    'Открываем схематический набор данных
    'для захвата метаданных пользователя
    Set rst = cnn.OpenSchema(adSchemaProviderSpecific, , _
        JET_SCHEMA_USERROSTER)
    'возвращаем текущих пользователей
    rst.MoveFirst
    Do Until rst.EOF
        ReturnUsers = rst(0) & ";" & ReturnUsers
        rst.MoveNext
    Loop

ExitHere:
```



```
rst.Close
Set rst = Nothing
cnn.Close
Set cnn = Nothing
Exit Function
HandleErr:
MsgBox "Ошибка " & Err.Number & ": " & _
Err.Description
Resume ExitHere
End Function
```

Как видите, программа потребовала определения константы JET_SCHEMA_USERROSTER — эта константа не встроена в библиотеку ADO.

Использование других наборов данных схем

Механизм баз данных Jet имеет ряд схематических наборов записей, отличных от списка пользователей. В табл. 19.1 приведены стандартные схематические наборы ADO, поддерживаемые механизмом Jet.

Таблица 19.1. Стандартные схематические наборы данных, поддерживаемые приложением Access

Константа	Описание наборов данных
adSchemaCheckConstraints	Правила проверки в базе данных
adSchemaColumns	Поля базы данных
adSchemaConstraintColumnUsage	Показывает, какие столбцы каким правилам подвержены
adSchemaForeignKeys	Вторичные ключи базы данных
adSchemaIndexes	Индексы базы данных
adSchemaKeyColumnUsage	Показывает, какие столбцы в какие ключи включены
adSchemaPrimaryKeys	Первичные ключи базы данных
adSchemaProcedures	Запросы, имеющие параметры
adSchemaProviderTypes	Список типов данных, поддерживаемый механизмом Jet
adSchemaReferentialConstraints	Правила целостности
adSchemaStatistics	Статистика базы данных
adSchemaTableConstraints	Правила уровня таблиц
adSchemaTables	Таблицы базы данных
adSchemaTrustees	Пользователи и группы базы данных
adSchemaViews	Запросы, не имеющие параметров

В дополнение к стандартным наборам данных схем механизм Jet, используемый приложением Access, определяет четыре собственных схематических набора данных, идентифицируемые следующими глобальными идентификаторами:

- {8703b612-5d43-11d1-bdbf-00c04fb92675} — статистика производительности;
- {947bb102-5d43-11d1-bdbf-00c04fb92675} — активные пользователи;
- {e2082df0-5d43-11d1-bdbf-00c04fb92675} — фильтры частей в репликах;
- {e2082df2-5d43-11d1-bdbf-00c04fb92675} — таблицы конфликтов в репликах.

В качестве примера использования наборов данных схем ниже приведена процедура, возвращающая некоторую информацию о таблицах текущей базы данных.

```
Public Sub ListTables()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim fld As ADODB.Field
    Dim i As Integer

    Set cnn = New ADODB.Connection
    On Error GoTo HandleErr

    'Открытие подключения к базе данных
    Set cnn = CurrentProject.Connection
    'Открытие набора записей схемы для извлечения метаданных
    Set rst = cnn.OpenSchema(adSchemaTables)

    'Цикл по всем записям набора данных
    Do Until rst.EOF
        ' Проход по всем полям строк
        For i = 0 To rst.Fields.Count - 1
            ' Печать имени поля и его значения
            Debug.Print rst.Fields(i).Name & ": " & rst(i)
        Next i
        ' Печать пустой строки после каждой записи
        Debug.Print
        rst.MoveNext
    Loop

ExitHere:
    rst.Close
    Set rst = Nothing
    cnn.Close
    Set cnn = Nothing
    Exit Sub
HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description
    Resume ExitHere
End Sub
```


Если запустить эту процедуру в базе данных примеров, вы получите список данных всех таблиц. В качестве примера приведены данные, извлеченные из таблицы `Clients`.

```
TABLE_SCHEMA:
TABLE_NAME: Clients
TABLE_TYPE: TABLE
TABLE_GUID:
DESCRIPTION:
TABLE_PROPID:
DATE_CREATED: 13.03.2004 11:30:57
DATE_MODIFIED: 15.05.2004 17:49:53
```

Примечание

Многие поля остались пустыми, потому что схематические наборы данных имеют общую направленность, и любое конкретное поле может быть просто неприменимым к конкретной базе данных. Определение полей во всех наборах данных схем содержится в *Приложении В* книги *OLE DB Programmer's Reference*, которое можно найти на узле http://msdn.microsoft.com/library/en-us/oledb/htm/oledbschema_rowsets.asp.

ПРАКТИКУМ

Использование события ошибки формы для разрешения конфликтов блокировки

При многопользовательском режиме работы с базой данных для разрешения конфликтов блокировки записей в наборе данных можно использовать собственную обработку ошибок. Однако во многих приложениях (в том числе и в базе данных примеров `TimeTrack`) пользователи взаимодействуют с формами, а не с наборами данных. Как же поступать в этом случае?

Обработку ошибок блокировки в формах можно выполнить двумя способами. Во-первых, вы можете разрешить обработку таких ошибок самому приложению `Access`; при этом будут отображаться сообщения, которые вы уже видели в настоящей главе на рис. 19.1 и 19.3.

Во-вторых, можете воспользоваться событием `Error` формы и написать собственную обработку ошибок. Указанное событие возникает при ошибках работы с данными в форме. В предлагаемом практикуме в модуль формы `Clients` добавлена следующая процедура, обрабатывающая ошибки блокировки записей (эта процедура уже добавлена в форму `Clients19`):

```
Private Sub Form_Error (DataErr As Integer, _
    Response As Integer)
    'Обработка ошибок многопользовательского режима
    On Error Goto HandleErr

    'Обработка ошибок на основе номера ошибки,
    'переданного приложением Access
    Select Case DataErr
        Case 7787 'Конфликт записи
```



```

        MsgBox "Другой пользователь изменил эту запись. " &
        "Щелкните ОК, чтобы увидеть его изменения.", vbCritical
        Response = acDataErrContinue
    Case 7788 'Данные изменены
        MsgBox "Другой пользователь изменил эту запись. " &
        "Щелкните ОК, чтобы увидеть его изменения.", vbCritical
        Response = acDataErrContinue
    Case Else
        'Остальные ошибки обрабатывает Access
        Response = acDataErrDisplay
    End Select

ExitHere:
    Exit Sub

HandleErr:
    MsgBox "При обработке ошибки возникла ошибка", vbCritical
    Resume ExitHere
End Sub

```

Процедура обработки события `Error` формы принимает два аргумента: номер ошибки, передаваемый приложением Access, вызвавшей данное событие, и отклик, позволяющий указать приложению способ обработки этой ошибки. В качестве второго аргумента вы можете вернуть Access следующие константы: `acDataErrContinue` указывает, что вы уже сами позаботились об ошибке, а `acDataErrDisplay` разрешает приложению принимать свои собственные меры (отображение сообщения об ошибке).

В данном случае мы перехватываем обе разновидности ошибки блокировки, предупреждаем пользователей, что данные будут обновлены, после чего продолжаем процесс. Access автоматически обновит данные в формы из источника данных после выполнения этой процедуры. Если возникает какая-либо другая ошибка данных, приложению Access предоставляется возможность выполнить ее обработку самостоятельно. В заключение, если в ходе обработки ошибки возникает какая-либо другая ошибка, пользователю выдается соответствующее сообщение и работа с формой прерывается. Последнее действие оградит вас от вполне вероятных бесконечных циклов возникновения ошибок.

Работа с файлами
данных

20

Понятие операции
файлового репозитория

В ЭТОЙ ЧАСТИ

Использование в Access расширенных средств VBA

20. Работа с файлами данных

21. Автоматизация других приложений

22. Работа с файлами XML

23. Использование Windows API

Работа с файлами данных

20

Понятие операций файлового ввода- вывода

Термин *операции файлового ввода-вывода* относится к функциям, позволяющим манипулировать содержимым файлов. Несмотря на то, что вам могут так никогда и не понадобиться эти функции в программировании, вы все равно ими неявно пользуетесь при работе со стандартными приложениями.

Процесс файлового ввода-вывода в языке VBA прост:

- для открытия файла используется функция `Open`;
- для извлечения содержимого файла используется функция `Input`;
- для записи информации в файл используется функция `Write`;
- для записи серии значений в открытый файл используется функция `Print`.

Перед тем как начать пользоваться функциями ввода-вывода, нужно усвоить понятие *метки файла*. В данном контексте меткой файла называют уникальное число, которое однозначно соответствует в операционной системе конкретному открытому файлу.

В ЭТОЙ ГЛАВЕ

Понятие операций файлового ввода-вывода	345
Открытие файлов	346
Чтение из файлов	349
Запись в файл	353
Печать в файл	354

Для открытия файла с целью ввода-вывода нужна неиспользованная метка. Это значение можно получить с помощью функции `FreeFile`, имеющей следующий синтаксис:

```
метка = FreeFile
```

где *метка* — переменная типа `Long Integer`.

Предупреждение

В главе 23, посвященной функциям Windows API, также будут использованы метки файлов. В то же время при работе с API нельзя использовать результат функции `FreeFile`.

Открытие файлов

Процесс работы с файлом начинается с его открытия. Для этого используют функцию `Open`, имеющую следующий синтаксис:

```
Open путь For режим [Access доступ] [блокировка] As [#] метка _  
[Len=длина_записи]
```

В табл. 20.1 перечислены аргументы этой функции.

Таблица 20.1. Аргументы функции `Open`

Аргумент	Описание
<i>путь</i>	Полный путь к открываемому файлу
<i>режим</i>	Append (Для добавления), Binary (Двоичный), Input (Для чтения информации), Output (Для записи) или Random (используется по умолчанию)
<i>доступ</i>	Read (Только для чтения), Write (Только для записи) или Read Write (Для чтения и записи)
<i>блокировка</i>	Shared (Разрешено совместное использование) (по умолчанию), Lock Read (Блокировка чтения), Lock Write (Блокировка записи) или Lock Read Write (Блокировка чтения и записи)
<i>метка</i>	Переменная, содержащая результат функции <code>FreeFile</code> . Это значение будет использоваться как идентификатор файла
<i>длина_записи</i>	При использовании режима Random это — длина одной записи; при использовании режимов Append или Input это — длина буфера в байтах

Аргумент *режим*

Аргумент *режим* определяет способ обработки системой Windows операции открытия файла, а также то, что вы с ним собираетесь делать впоследствии. Большей частью вам потребуется совершать следующие операции с файлом:

- *Просмотр содержимого файла без его редактирования или добавления информации.* Используйте режим `Input` — при этом будет открываться доступ только для чтения.
- *Добавление и редактирование информации.* Используйте режим `Append` — он открывает возможность дозаписи в конец файла.
- *Удаление старого файла и создание нового под тем же именем.* Используйте режим `Output`.
- *Добавление или редактирование значений в двоичном режиме* (обычно его используют при работе с графическими файлами). Используйте режим `Binary`.
- *Добавление и редактирование значений, зависящих от конкретной длины записи.* Используйте режим `Random` совместно с аргументом `длина_записи`.

Аргумент *доступ*

Этот аргумент может быть установлен в три значения, которые говорят сами за себя: `Read`, `Write` и `Read Write`. Не забывайте, что кроме передаваемых в этом аргументе характеристик доступа к файлу в самой операционной системе должны присутствовать соответствующие разрешения для вашей учетной записи. К примеру, если вы пытаетесь открыть файл с уровнем разрешений для записи (`Write`), а в системе безопасности Windows вам разрешено только его чтение, запрос на открытие приведет к ошибке. По этой причине не рекомендуется в фрагменты программ работы с файлами закладывать собственную обработку ошибок.

Аргумент *блокировка*

В однопользовательских приложениях блокировка не нужна. В противоположность этому, в многопользовательских приложениях по умолчанию разрешается совместное использование файла (значение `Shared`). Это значит, что в момент, когда у вас открыт данный файл, другие пользователи имеют также возможность работы с ним для чтения и записи. Ниже представлено краткое описание допустимых режимов блокировки:

- `Shared`. Пользователи с соответствующими правами доступа могут извлекать значения из файла и добавлять в него новые, независимо от того, кем этот файл открыт.
- `Lock Read`. Пользователи с соответствующими правами доступа могут записывать в файл данные, однако не могут извлекать их, если файл открыт кем-то другим.
- `Lock Write`. Пользователи с соответствующими правами доступа могут извлекать из файла данные, однако не могут записывать их, если файл открыт кем-либо другим.

- **Lock Read Write.** Если файл кем-то открыт, другие пользователи не могут извлекать из него информацию и производить в него запись.

Простой пример открытия файла

Приведенной ниже процедуре в качестве аргумента передается путь к файлу, после чего он открывается функцией `Open`.

```
Sub OpenFile(fil As String)
    'Открытие файла
    Dim hFile As Long
    hFile = FreeFile
    Open fil For Input Access Read Shared As hFile
    MsgBox fil & " = " & hFile
    Close hFile
End Sub
```

На самом деле в этой процедуре не выполняется ничего, кроме открытия файла, имя которого было передано в качестве аргумента. Файл открывается только для чтения и к нему разрешен совместный доступ. Это значит, что из файла вы можете извлекать данные, но не сможете производить в него запись. В то же время все остальные пользователи (естественно, с соответствующими разрешениями) могут читать этот файл и записывать в него данные.

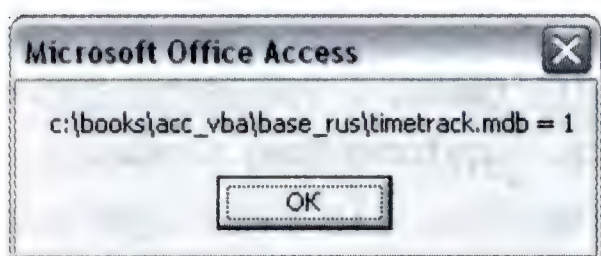


Рис. 20.1. Теперь вы знаете метку открытого файла

Вы можете протестировать эту процедуру, взяв ее из модуля `Chapter20` базы данных примеров `TimeTrack`, или ввести ее вручную в стандартный модуль. В окне `Immediate` введите следующую инструкцию (дополнив ее соответствующим путем к файлу):

`OpenFile "путь\TimeTrack.mdb"`

После открытия файла `TimeTrack.mdb` процедура откроет диалоговое окно, аналогичное показанному на рис. 20.1. В нем отображается имя открытого файла и ассоциированная с ним метка. Щелкните на кнопке `OK` и закройте окно.

Предупреждение

Когда вся работа с файлом проделана, не забывайте его закрывать инструкцией `Close`; в противном случае он может так и остаться заблокированным для других пользователей.

Несмотря на то, что метку файла достаточно легко распознать, в явном виде этой меткой пользоваться не рекомендуется. На практике лучше обращаться к переменной, содержащей эту метку, — тогда вам не придется ее запоминать.

Примечание

При одновременной работе более чем с одним файлом открывайте файл с использованием метки, возвращенной функцией `FreeFile`, а затем уже вызывайте эту функцию для получения новой свободной метки; в противном случае функция `FreeFile` будет все время возвращать одну и ту же метку для разных файлов (пока какой-нибудь из них не будет реально открыт). Если неправильно скомпоновать инструкции получения меток и открытия файлов, вы, сами того не зная, можете получить несколько меток, имеющих одно и то же значение. А последовательное открытие ряда файлов с одной и той же меткой (без предварительного закрытия предыдущего), естественно, приведет к ошибке.

Чтение из файлов

Под термином *чтение* обычно подразумевают операцию извлечения информации. При открытии файла в режиме `Input`, `Output` или `Append` можно использовать следующие инструкции чтения:

- `Input;`
- `Line Input #;`
- `Input #.`

Функция `Input`

Простейшим способом извлечения данных является использование функции `Input`, имеющей следующий синтаксис:

`Input (число, [#]метка)`

где *число* определяет количество извлекаемых символов, а *метка* является любой допустимой меткой файла. Функция `Input` возвращает все прочитанные символы, в том числе символы возврата каретки, перевода строки и пробелов.

Рассмотрим простую процедуру, использующую функцию `Input` для извлечения данных из экспортированной таблицы. Естественно, в первую очередь нам нужно экспортировать таблицу. Для этого мы выберем в меню команду **Файл⇒Экспорт (File⇒Export)** и экспортируем таблицу `Employees` в текстовый файл (при экспорте оставьте все настройки, принятые по умолчанию, без изменений). После этого введите приведенную ниже процедуру в стандартный модуль или воспользуйтесь готовой, содержащейся в модуле `Chapter20` базы данных `TimeTrack`.

```
Sub ReadInput(fil As String)
    'Вывод данных из переданного файла
    'в окно Immediate
    Dim hFile As Long
    hFile = FreeFile
    Open fil For Input Access Read Shared As hFile
    Debug.Print Input(LOF(hFile), hFile)
    Close hFile
End Sub
```


Теперь выполните следующую инструкцию в окне Immediate (не забыв подставить правильный путь к файлу экспорта):

```
ReadInput "путь\employees.txt"
```

Компонент LOF (hFile) в функции Input определяет длину файла, в результате чего функция возвращает все содержимое файла (рис. 20.2).

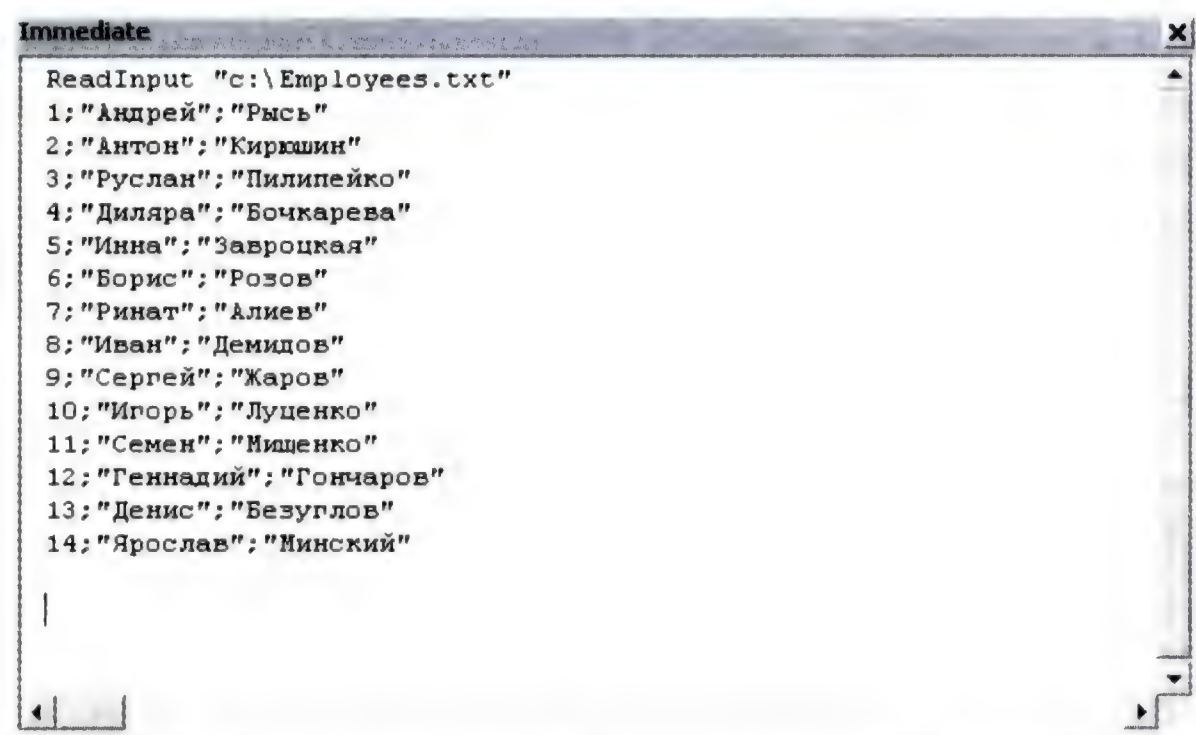


Рис. 20.2. Функция Input использована для извлечения всего содержимого файла

Примечание

Экспортированный файл employees.txt содержал переводы строк. Мастер экспорта приложения Access вставлял их при экспорте данных между записями. Именно поэтому данные, выведенные в окно Immediate, оказались хорошо организованными: один сотрудник — одна строка. Если в файле разрывов строк не содержится, функция Input извлечет его содержимое в одну длинную строку, разорванную в месте, соответствующем установленным соглашениям контейнера.

EOF и LOF

В приведенном выше примере мы познакомились с функцией LOF. Каждый раз, когда с содержимым файла нужно что-либо сделать, вам несомненно придется пользоваться функциями LOF или EOF.

Функция EOF помогает определить конец файла. С технической стороны, эта функция возвращает булево значение, указывающее на то, что положение текущего байта находится в конце файла (значение True, если конец файла достигнут, или False — в противном случае).

Термин *положение текущего байта*, использованный в предыдущем предложении, подразумевает текущий указатель в файле. При открытии файла его значение устанавливается в нуль. Это положение вы можете себе представить как метку перед самым первым байтом файла (как правило, байт соответствует одному символу, в том числе и пробелу).

Функция LOF возвращает размер открытого файла в байтах. В предыдущем примере это значение использовалось для указания функции Input длины извлекаемого содержания файла.

Инструкция Line Input

Подобно функции Input, инструкция Line Input # читает данные из открытого файла, однако по одной строке за один вызов. Эта инструкция имеет следующий синтаксис:

Line Input #метка, переменная

где *метка* является меткой файла, а *переменная* — это переменная, в которую сохраняются извлеченные данные.

Следующая процедура извлекает содержимое файла экспорта практически так же, как это делалось в предыдущем примере с помощью функции Input.

```
Sub ReadLineInput(fil As String)
    'Вывод данных из переданного файла
    'в окно Immediate
    Dim hFile As Long
    Dim strLine As String
    hFile = FreeFile
    Open fil For Input Access Read Shared As hFile
    Do Until EOF(hFile)
        Line Input #hFile, strLine
        Debug.Print strLine
    Loop
    Close hFile
End Sub
```

В окне Immediate введите следующую инструкцию (не забыв указать правильный путь):

```
ReadLineInput "путь\employees.txt"
```

Теперь в функции данные извлекаются не как одна сплошная строка (как в случае использования функции Input), а построчно, в цикле Do Until. Если удалить цикл Do Until, инструкция Debug.Print выведет на печать только первую строку.

Инструкция Input

Инструкция Input # практически во всем идентична инструкции Line Input #, однако она более гибкая. Эта инструкция извлекает данные и присваивает их некоторой переменной. Она имеет следующий синтаксис:

Input #метка, список_переменных

где *список_переменных* — это список переменных, разделенных запятыми. Каждый элемент данных файла сохраняется функцией Input # в отдельную переменную. К тому же эта функция удаляет кавычки, ограничивающие строковые значения, и преобразует строки дат в даты VBA.

Приведенная ниже процедура возвращает те же данные, что и предыдущие две, однако в этом случае данные присваиваются трем переменным: `strID`, `strFN` и `strLN`, которые при выводе функцией `Debug.Print` объединяются.

```
Sub ReadInputPoundSign(fil As String)
    'Вывод данных из переданного файла
    'в окно Immediate
    Dim hFile As Long
    Dim strID As String
    Dim strFN As String
    Dim strLN As String
    hFile = FreeFile
    Open fil For Input Access Read Shared As hFile
    Do Until EOF(hFile)
        Input #hFile, strID, strFN, strLN
        Debug.Print strID & vbTab & strFN & " " & strLN
    Loop
    Close hFile
End Sub
```

В окне **Immediate** введите следующую инструкцию (не забыв указать правильный путь)¹:

```
ReadInputPoundSign "путь\employees.txt"
```

Выведенные в окно **Immediate** данные будут иметь тот же состав, что и в предыдущих примерах, однако форматирование будет другим (рис. 20.3).

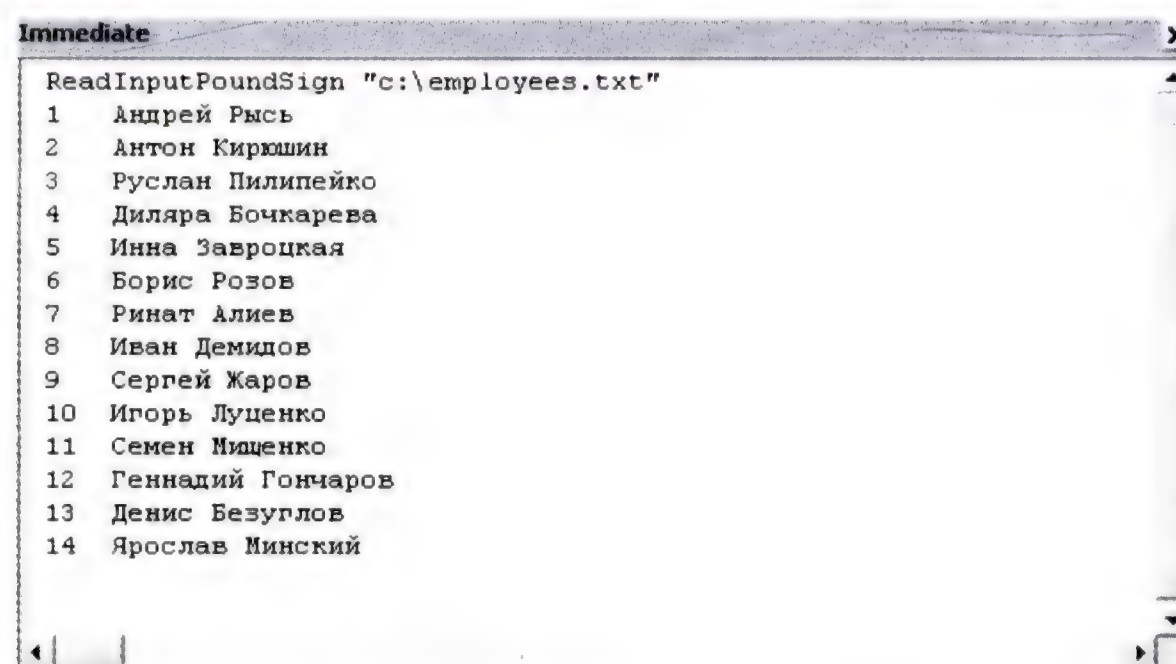


Рис. 20.3. Для управления возвращенными данными используйте функцию `Input #`

¹ При работе с русской версией операционной системы Windows XP придется выполнить дополнительный экспорт таблицы `Employees` в текстовый файл, на этот раз в качестве разделителя данных в строке использовав символ запятой. Однако чтобы такой экспорт сработал без ошибок, в региональных настройках следует предварительно изменить символ разделителя дробной и целой частей с запятой на точку. (Не забудьте после работы с примером вернуть региональные настройки системы назад.) — *Примеч. ред.*

Запись в файл

Записью в файл называют процесс добавления в файл значений. При открытии файла в режиме Input, Output или Append используются инструкции Print # или Write #; при открытии же в режиме Random или Binary — инструкция Put.

Инструкция Write # имеет следующий синтаксис:

```
Write #метка[, список_вывода]
```

где *список_вывода* — это одна или несколько разделенных запятыми переменных или констант, значения которых вы хотите добавить в файл. Эти значения можно разделять запятыми, пробелами или точками с запятой. Если аргумент *список_вывода* опустить, а после аргумента *метка* поставить запятую, в файл будет выведена пустая строка.

Если не учесть ряд перечисленных ниже особенностей этой инструкции, могут возникнуть проблемы:

- Все числовые данные записываются в файл с десятичным разделителем в виде точки.
- Все булевы значения записываются как #TRUE# или #FALSE#.
- Все даты записываются с помощью универсального формата дат.
- Если *список_вывода* пуст, в файл ничего не записывается.
- Если аргумент *список_вывода* равен значению Null, в файл выводится #NULL#.
- Значения ошибок записываются в виде #ERROR код_ошибки#.

Следующая процедура позволяет добавить в файл одну строку и вывести файл на печать.

```
Sub InputAndWrite(fil As String, _
  id As Long, fn As String, ln As String)
  'Запись в файл и последующий вывод данных
  'в окно Immediate.
  Dim hFile As Long
  Dim strID As String
  Dim strFN As String
  Dim strLN As String
  hFile = FreeFile
  Open fil For Append Access Write As hFile
  Write #hFile, id, fn, ln
  Close hFile
  hFile = FreeFile
  Open fil For Input Access Read Shared As hFile
  Do Until EOF(hFile)
    Input #hFile, strID, strFN, strLN
    Debug.Print strID & vbTab & strFN & " " & strLN
  Loop
  Close hFile
End Sub
```


Выполните нижеуказанную инструкцию в окне Immediate; при этом в файл `employees.txt` будет добавлен сотрудник под номером 15 — Василий Пупкин²:

```
InputAndWrite "c:\employees.txt", 15, "Василий", "Пупкин"
```

Результат выполнения этой инструкции показан на рис. 20.4. В списке указан новый сотрудник под номером 15. Инструкция `Write #` использовала переданные процедуре аргументы — `id`, `fn` и `ln` — и скомпоновала из них новые данные. Обратите внимание, что при открытии файла для записи инструкция `Open` несколько отличается. В данном случае использован режим `Append` (добавления данных) и тип доступа `Write` (только для записи). Для создания нового файла (который перепишет поверх уже существующий) при открытии файла вместо режима `Append` нужно указывать режим `Output`.

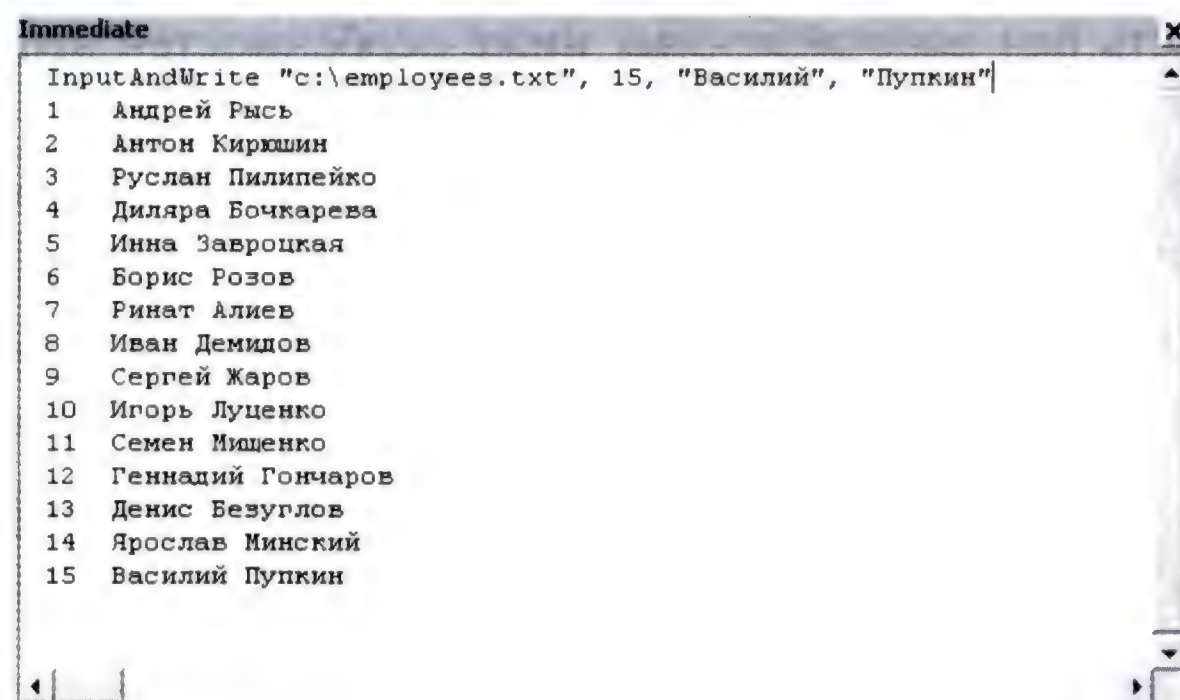


Рис. 20.4. Использование функций ввода-вывода для добавления новых данных

Печать в файл

При использовании для вывода в файл инструкции `Print #` применяют следующий синтаксис:

```
Print метка[, СПИСОК_ВЫВОДА]
```

Информация, передаваемая данной инструкции, в точности совпадает с той, которую мы рассматривали для инструкции `Write #`. Все различие между этими инструкциями состоит в том, что `Write #` производит вывод в уже открытый файл, а `Print #` создает новый.

Для добавления пробелов в `СПИСОК_ВЫВОДА` используют функцию `Spc(n)`, где `n` — количество вставляемых пробелов. Аналогично для вставки

² При работе с русской версией операционной системы Windows XP предполагается, что экспорт файла, как и в предыдущем примере, выполнен с использованием запятой в качестве разделителя, а в региональных настройках дробную и целую части числа разделяет точка. — Примеч. ред.

символов табуляции используют функцию `Tab(n)`, где n представляет абсолютный номер столбца.

В следующей функции считывается содержимое всего файла, после чего создается новый файл печати.

```
Sub PrintPoundSign(filSource As String, filPrint As String)
    'Печать данных из передаваемого файла
    'в файл печати.
    Dim hFileSource As Long
    Dim hFilePrint As Long
    Dim strLine As String
    hFileSource = FreeFile
    Open filSource For Input Access Read Shared As hFileSource
    hFilePrint = FreeFile
    Open filPrint For Output Access Write As hFilePrint
    Do Until EOF(hFileSource)
        Line Input #hFileSource, strLine
        Print #hFilePrint, strLine
    Loop
    Close hFileSource
    Close hFilePrint
End Sub
```

Для запуска этой процедуры введите в окне Immediate следующую инструкцию:

```
PrintPoundSign "c:\employees.txt", "PrintFile"
```

С помощью текстового процессора Word откройте новосозданный файл с именем `PrintFile` (он должен находиться в каталоге, используемом приложением Access по умолчанию). Его содержимое в точности будет совпадать с содержимым файла `employees.txt` (рис. 20.5).

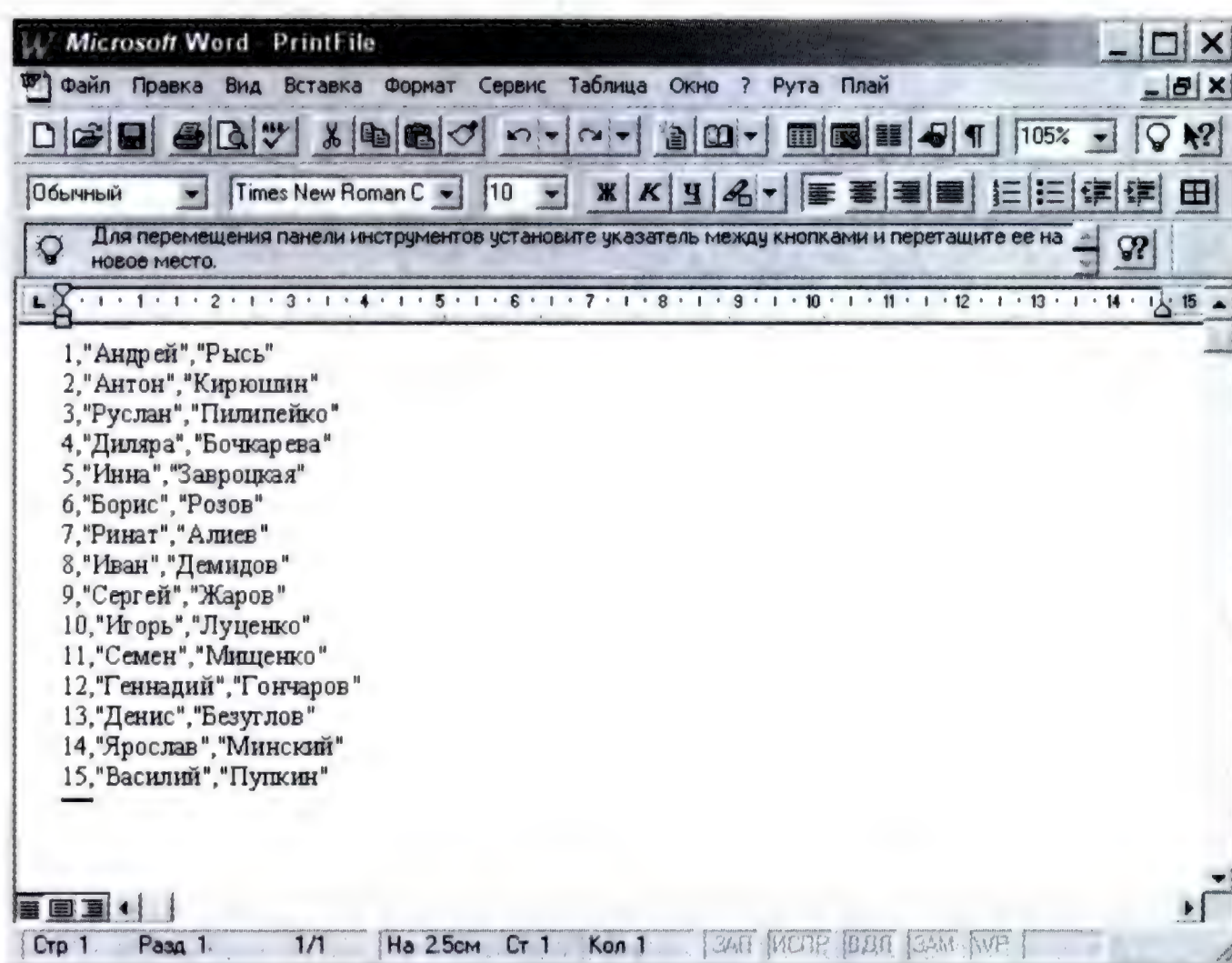


Рис. 20.5. Содержимое файла печати, идентичное исходному файлу

ПРАКТИКУМ**Использование операций ввода-вывода для нумерации строк в текстовом файле**

Пользователям иногда требуется экспортировать информацию о клиентах в текстовый файл для обсуждения некоторых вопросов с другими сотрудниками. Однако при этом желательно пронумеровать строки в тексте. Естественно, можно открыть файл экспорта и пронумеровать строки вручную, но лучше автоматизировать этот процесс с помощью функций ввода-вывода.

Для начала откроем форму Switchboard и добавим на нее новую командную кнопку. Присвоим ей имя cmdExport и зададим подпись Экспорт в текстовый файл. После этого в модуль формы добавим процедуру обработки события Нажатие кнопки созданного элемента:

```
Private Sub cmdExport_Click()
    'Экспорт таблицы Clients
    'в текстовый файл с разделителями
    On Error GoTo HandleErr
    DoCmd.TransferText acExportDelim, , _
        "Clients", "Clients.txt"
    Call NumberClientList
    Exit Sub

HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description
End Sub
```

Эта процедура достаточно проста — она экспортирует всю таблицу Clients в файл с именем Clients.txt, создаваемый в текущей директории (при необходимости вы можете к имени файла добавить полный путь).

Для создания функции NumberClientList, вызываемой обработчиком события Нажатие кнопки, откройте стандартный модуль и введите следующий фрагмент программы (можете также использовать готовую процедуру, находящуюся в модуле Chapter20 базы данных примеров):

```
Public Sub NumberClientList()
    'Добавление номеров строк
    'в текстовый файл списка клиентов.
    Dim hFileSource As Long
    Dim hFilePrint As Long
    Dim strInput As String
    Dim i As Integer
    On Error GoTo HandleErr
    hFileSource = FreeFile
    Open "Clients.txt" For Input Access Read As hFileSource
    hFilePrint = FreeFile
    Open "ClientsPrint.txt" For Output Access Write As hFilePrint
    Do Until EOF(hFileSource)
        i = i + 1
        Line Input #hFileSource, strInput
        Print #hFilePrint, i, strInput
    Loop
```



```
Exit Sub

ExitHere:
    Close hFileSource
    Close hFilePrint
    Exit Sub
HandleErr:
    MsgBox "Ошибка " & Err.Number & ": " & _
        Err.Description
    Resume ExitHere
End Sub
```

Сохраните этот модуль и закройте его. Откройте измененную форму Switchboard (рис. 20.6) и щелкните на новой командной кнопке. После экспорта таблицы Clients в текстовый файл процедура обработки события нажатия кнопки вызывает функцию NumberClientList, которая этот файл открывает и делает в цикле Do Until следующее:

- последовательно извлекает каждую из строк файла функцией Line Input;
- выполняет инструкцию `i = i + 1`, увеличивая счетчик цикла;
- с помощью функции Print выводит в новый файл строки исходного файла, дополненные в начале текущим номером строки `i`.

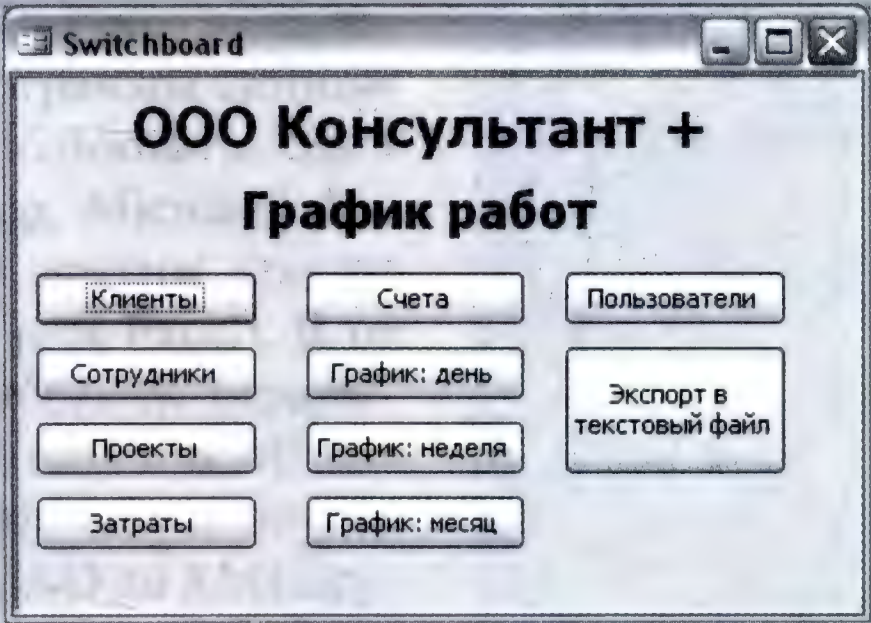


Рис. 20.6. Для создания текстового файла щелкните на кнопке Экспорт в текстовый файл

На рис. 20.7 показан новый текстовый файл с пронумерованными строками (в данном случае номера строк совпадают с идентификационными номерами клиентов).

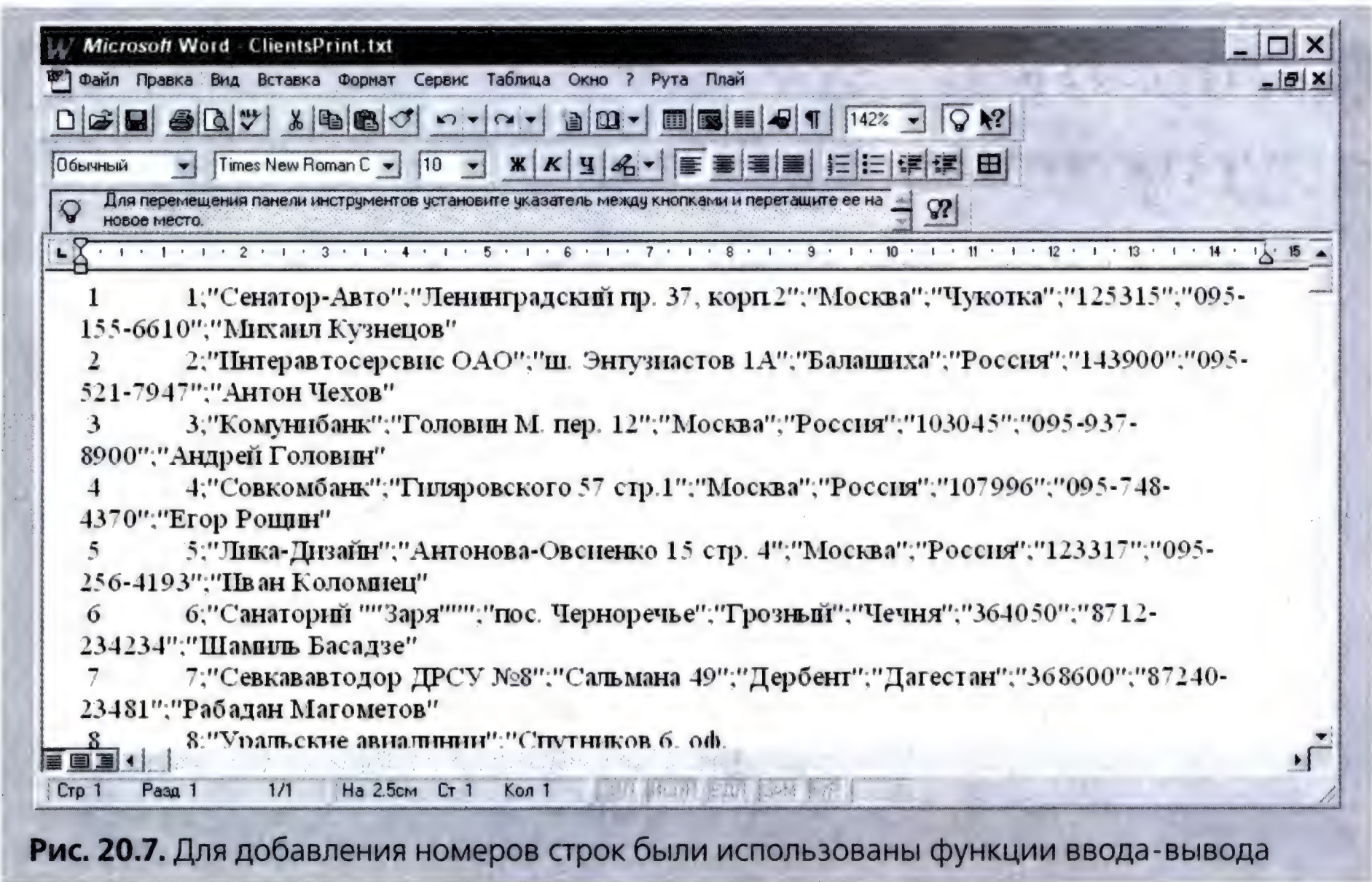


Рис. 20.7. Для добавления номеров строк были использованы функции ввода-вывода

Автоматизация других приложений

21

Понятие автоматизации

Технологии, которыми мы пользовались в настоящей книге до сих пор, были основаны главным образом на возможностях приложения Microsoft Access. Это приложение, как вы знаете, является составной частью пакета Microsoft Office — группы продуктов, созданной для совместного использования. Именно здесь на сцену снова выходит понятие *автоматизации*. Создавая программы автоматизации, можно использовать одно приложение (например, Microsoft Access) для управления другим (таким как Word, PowerPoint или Excel). В то же время эта способность не ограничивается одним лишь пакетом Microsoft Office. Существуют сотни других приложений, от AutoCAD до XMLSpy, которыми можно управлять посредством автоматизации.

Программы автоматизации всегда работают с двумя приложениями: *клиентским* и *серверным*. Клиентским называют приложение, в котором запускается на выполнение код VBA (в нашем случае это Access); серверным же — приложение, которое предоставляет функциональность программе автоматизации. Клиент создает один или несколько объектов из сервера, а после этого использует свойства и методы этих объектов для выполнения собственных задач.

В ЭТОЙ ГЛАВЕ

Понятие автоматизации.....	359
Создание ссылок на объекты ...	360
Создание объектов на сервере автоматизации	362
Взаимодействие с приложением Excel из Access	365
Взаимодействие с приложением Word из Access	367

К примеру, приложение Microsoft Access может использовать автоматизацию с приложением Microsoft Word в качестве сервера для создания нового документа Word. Создав такой документ (т.е. новый объект `Word.Document`), программа Access получает доступ ко множеству его свойств добавления и форматирования текста, а также сохранения с конкретным именем файла. Результатом станет создание файла документа Word (с расширением `.doc`) из среды приложения Access.

Автоматизация не переносит объекты из одного приложения в другое. Вместо этого она позволяет одному приложению управлять объектами, созданными другим приложением. И в этом заключено их главное различие. К примеру, если вы используете Microsoft Word в качестве сервера автоматизации, то пользователь должен иметь на своем компьютере это приложение установленным. В то же время пользователю не придется запускать приложение и манипулировать им вручную — за него это сделает программа автоматизации.

Предупреждение

Если существует вероятность того, что пользователь попытается запустить программу автоматизации в системе, в которой отсутствует требуемое серверное приложение, обязательно включите в программу обработку ошибок, предотвращающую краш выполнения программы при возникновении подобных нестандартных ситуаций.

Создание ссылок на объекты

Клиентам автоматизации нужен способ определения того, какие именно объекты конкретного сервера автоматизации они могут использовать в своей работе. Эта информация хранится в файлах, называемых *библиотеками типов*, обычно имеющих расширения `.tlb` или `.olb`. Библиотеки типов содержат информацию об:

- объектах;
- методах;
- свойствах;
- событиях.

Несмотря на то, что для использования объектов из сервера автоматизации совсем не требуется обязательное наличие библиотеки типов, она станет неплохим подспорьем. Подумайте о том, что при отсутствии библиотеки типов при написании программ автоматизации вы не сможете воспользоваться средствами *IntelliSense* и *Object Browser*. Поэтому рекомендуется всегда использовать библиотеку типов. Для использования же библиотеки типов при автоматизации на них нужно установить ссылки. Вы узнаете, как это выполняется, изучив примеры библиотек типов приложений Microsoft Word и Microsoft Excel, которыми мы будем пользоваться в настоящей главе.

1. Откройте редактор VBE, нажав в Access комбинацию клавиш <Alt+F11>.
2. Выберите в меню пункт Tools⇒References. Откроется диалоговое окно ссылок, показанное на рис. 21.1.

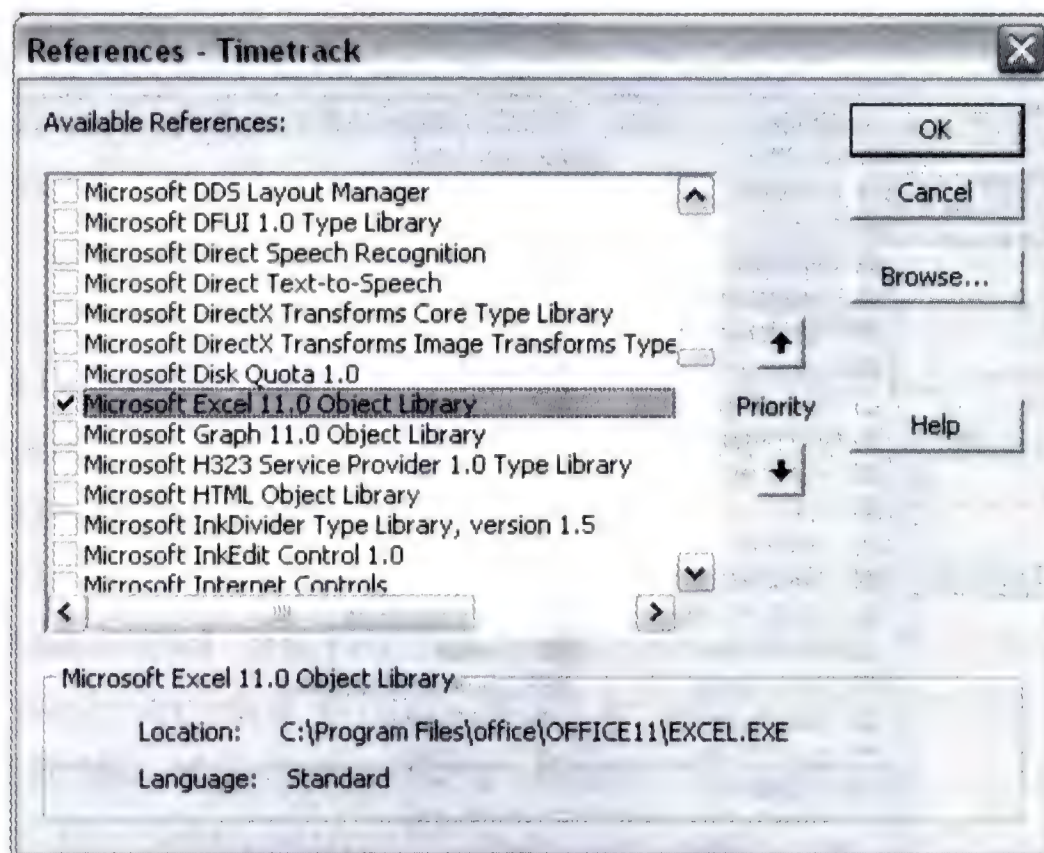


Рис. 21.1. Добавление ссылок на библиотеки типов

3. Найдите в списке пункты Microsoft Excel 11.0 Object Library и Microsoft Word 11.0 Object Library и установите на них флажки.
4. Щелкните на кнопке ОК, и ссылки на эти библиотеки типов будут добавлены в текущую базу данных Access.

После того как ссылки на библиотеки типов добавлены, вы сможете найти их объекты в Обозревателе объектов. Чтобы запустить этот обозреватель, нажмите клавишу <F2>. После этого выберите нужную библиотеку типов в комбинированном списке Project/Library. На рис. 21.2 показаны несколько объектов библиотеки типов приложения Excel. Как видите, эти объекты работают с ячейками и графиками электронных таблиц — т.е. с объектами, чуждыми базе данных Access, в которой мы находимся.

Примечание

Описанный только что процесс ничем не отличается от использования обозревателя для объектов самого приложения Access. Единственным отличием является то, что ссылку на собственную библиотеку типов приложение Access устанавливает автоматически.

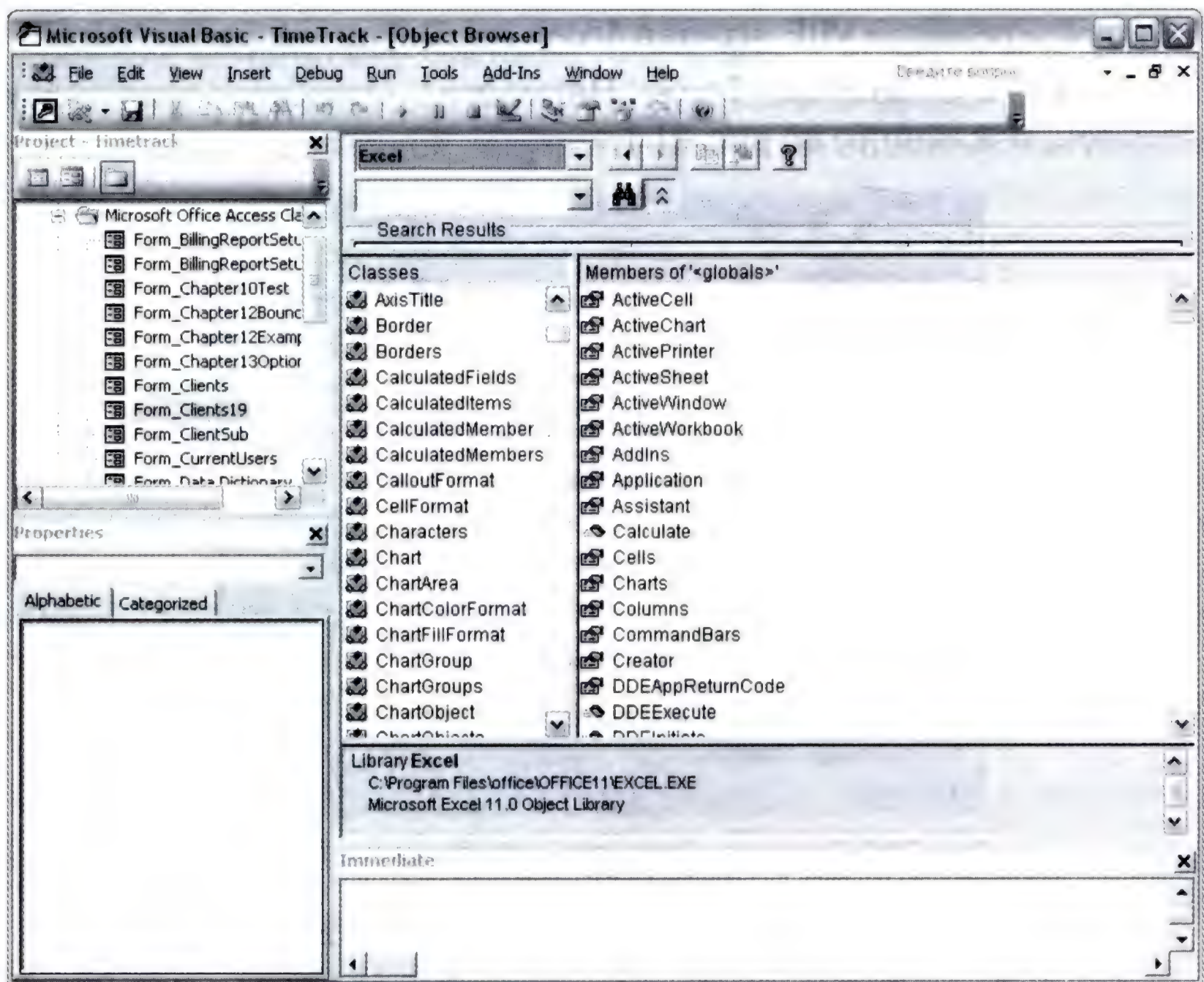


Рис. 21.2. С помощью Обозревателя объектов можно инспектировать библиотеки типов

Создание объектов на сервере автоматизации

Для работы с объектами из другого приложения их нужно вначале создать в тексте программы Access. Существует три способа реализовать эту операцию:

- использование функции `CreateObject`;
- использование функции `GetObject`;
- использование раннего связывания.

Рассмотрим все эти три способа более подробно.

Использование функции `CreateObject`

Функция `CreateObject` позволяет определить объект по его имени и создать один его экземпляр. Вот краткий пример ее использования:

```
Sub CreateObjectDemo()  
    ' Использование Word в качестве сервера автоматизации  
    ' с помощью функции CreateObject  
    Dim objWord As Object  
    Set objWord = CreateObject("Word.Application")  
    objWord.Visible = True  
    objWord.Quit  
End Sub
```


В данном примере для ссылки на создаваемый объект используется переменная специального типа `Object`. Это тип переменных, которые могут ссылаться на любые объекты: на документ Word, на диаграмму Excel, на форму Access.

Функция `CreateObject` преобразовывает обобщенную переменную объекта в конкретный тип объекта. Аргументом функции `CreateObject` является то, что известно под именем `ProgID` (идентификатор объекта). Это — уникальное значение, сопоставленное с типом создаваемого объекта. Большинство идентификаторов объекта можно получить, объединив имя приложения с именем его объекта. Так, идентификатор `Word.Application` соответствует объекту `Application` приложения Word (а не собственному объекту `Application` приложения Access).

После того как строка программы с вызовом функции `CreateObject` отработала, переменная `objWord` уже ссылается на экземпляр объекта `Application` приложения Word. Этот объект представляет собой сессию работы с Microsoft Word (равно как объект `Application` приложения Access представляет сессию работы с последним). Однако если вы установите на этой строке контрольную точку и остановитесь в этом месте, вы не найдете нигде запущенного приложения Word. Это происходит потому, что Word, равно как и большинство других серверов автоматизации, запускается в невидимом режиме при вызове из программы автоматизации. В следующей строке программы свойству видимости приложения Word присваивается значение `True`, в результате чего окно этого приложения отображается на экране.

И наконец, вызывается метод `Quit` объекта `Application` приложения Word. Это — эквивалент вызова команды меню Файл⇒Выход (`File⇒Exit`).

Хотелось бы заметить, что кроме новой функции `CreateObject` в данном примере отсутствует какой-либо новый синтаксис. После того как мы извлекли объект автоматизации, можно использовать его свойства и методы точно так же, как и в любом объекте Access.

Использование функции `GetObject`

Функция `GetObject` очень близка к функции `CreateObject`. Вот пример ее использования:

```
Sub GetObjectDemo()
    ' Использование Word в качестве сервера автоматизации
    ' с помощью функции GetObject
    Dim objWord As Object
    Set objWord = GetObject("", "Word.Application")
    objWord.Visible = True
    objWord.Quit
End Sub
```

Если запустить на выполнение эту процедуру, то вы обнаружите, что она работает в точности так же, как и `CreateObjectDemo`. В то же время, меж-

ду функциями `CreateObject` и `GetObject` отмечается два существенных различия:

- Функция `CreateObject` всегда создает новый экземпляр сервера автоматизации. Функция `GetObject` может использовать тот экземпляр, который уже запущен в системе.
- Функция `GetObject` имеет первый, необязательный параметр, задающий документ. Если вызвать ее следующим образом: `GetObject("c:\temp\doc1.doc", "Word.Application")`, то при необходимости будет создан экземпляр объекта `Application`, и в это приложение будет загружен документ, находящийся в заданном файле.

Функцию `GetObject` лучше использовать тогда, когда нужно работать с определенным документом или когда вы столкнулись с проблемой минимизации влияния приложения автоматизации на задействованные в системе ресурсы (это больше относится к компьютерным системам с ограниченными ресурсами памяти), так как в данном случае не запускается несколько дополнительных копий приложения сервера автоматизации.

Использование раннего связывания

Функции `CreateObject` и `GetObject` используют то, что в программировании называют *поздним связыванием*. При позднем связывании в тексте программы явно не указывается, какой тип объекта будет использоваться. Вместо этого используется объект общего типа `Object`, а интерпретатор VBA в ходе выполнения программы уже сам решает, какой тип объекта создавать. Третьим способом создания объектов является использование *раннего связывания*, которое позволяет применять родные типы переменных сервера автоматизации непосредственно в коде VBA.

Совет

Позднее связывание может работать как со ссылкой на библиотеку типов сервера автоматизации, так и без нее; для раннего же связывания такая ссылка обязательна. Несмотря на то, что язык VBA позволяет добавлять ссылку на библиотеку типов в тексте программы (с помощью коллекции `References`), опыт подсказывает, что на практике это срабатывает не всегда.

Далее представлен пример создания объекта, преобразованный в метод раннего связывания:

```
Sub EarlyBindingDemo()
    ' Использование Word в качестве сервера автоматизации
    ' с помощью раннего связывания
    Dim objWord As Word.Application
    Set objWord = New Word.Application
    objWord.Visible = True
    objWord.Quit
End Sub
```


При раннем связывании в коде VBA используют родной тип объекта. Этот режим становится возможным благодаря установленной ссылке на библиотеку типов сервера автоматизации. После того как ссылка установлена, вы можете объявлять и инициализировать объект автоматизации в точности так же, как и любой собственный объект Access.

Рекомендуется использовать раннее связывание по нескольким причинам.

- Вам не придется запоминать два разных синтаксиса создания объектов.
- Вы получите все преимущества механизма IntelliSense при написании программы, а это значит, что вы сможете выявить ошибки еще до того, как их обнаружат пользователи программы.
- Раннее связывание работает быстрее функций `CreateObject` и `GetObject`.

Взаимодействие с приложением Excel из Access

Как вы уже убедились, синтаксис программ автоматизации не так уж сложно изучить. Вы просто устанавливаете соответствующую ссылку, а затем используете объекты сервера автоматизации так же, как и родные объекты Access. Реальная сложность автоматизации скрывается в необходимости изучения возможностей объектов другого приложения. Если на вашем компьютере установлена масса разнообразных приложений, вам потенциально доступны десятки, а может даже и сотни серверов автоматизации.

В одной главе невозможно предоставить исчерпывающий обзор средств автоматизации работы всех этих приложений. Вместо этого мы изучим несколько примеров: в одном из них будет задействовано приложение Microsoft Excel, в другом — Microsoft Word.

Совет

Приложения пакета Microsoft Office делают все свои функции доступными другим программам через средства автоматизации, поэтому такой выбор послужит хорошим стартом.

В первом примере будет показано, как результаты запроса `WeeklyTimeslips` передать в Excel. Перемещение данных между разными приложениями с целью последующего анализа чаще всего связано с использованием автоматизации. Приложение Excel значительно упрощает получение итоговых значений для групп записей, одновременно позволяя обозревать и детали, что достаточно сложно реализовать в Access.

Для выполнения экспорта данных в Excel вы можете открыть модуль `Chapter21` в базе данных примеров и запустить процедуру `QueryToExcel` из окна `Immediate`. Вот ее текст:

```
Sub QueryToExcel()  
    ' Отправляет результаты запроса WeeklyTimeslips
```



```
' в Excel для дальнейшего анализа

Dim rstWeeklyTimeslips As ADODB.Recordset
Dim objXL As Excel.Application
Dim objWS As Excel.Worksheet
Dim fld As ADODB.Field
Dim intCol As Integer
Dim intRow As Integer

Set rstWeeklyTimeslips = New ADODB.Recordset

' Помещаем нужные данные в набор записей
rstWeeklyTimeslips.Open _
    "WeeklyTimeslips", CurrentProject.Connection

' Запускаем Excel
Set objXL = New Excel.Application
' Создаем новый рабочий лист
objXL.Workbooks.Add
Set objWS = objXL.ActiveSheet

' Копируем данные
' Вначале имена полей
For intCol = 0 To rstWeeklyTimeslips.Fields.Count - 1
    Set fld = rstWeeklyTimeslips.Fields(intCol)
    objWS.Cells(1, intCol + 1) = fld.Name
Next intCol
' Затем фактические данные
intRow = 2
Do Until rstWeeklyTimeslips.EOF
    For intCol = 0 To rstWeeklyTimeslips.Fields.Count - 1
        objWS.Cells(intRow, intCol + 1) = _
            rstWeeklyTimeslips.Fields(intCol).Value
    Next intCol
    rstWeeklyTimeslips.MoveNext
    intRow = intRow + 1
Loop

' Делаем рабочий лист видимым
objXL.Visible = True

' Метод Excel.Quit мы не вызываем, поскольку
' рабочий лист должен быть доступным
' после завершения процедуры
End Sub
```

Эта процедура может на первый взгляд показаться сложной, поскольку в процедуре приходится одновременно манипулировать объектами Excel и ADO. После объявления переменных мы начинаем свою работу с открытия набора данных, созданных запросом `WeeklyTimeslips`. Мы не утруждали себя установкой типа блокировки и курсора в этом наборе данных, поэтому открыли его как набор с последовательным доступом и доступный только для чтения. Это — самый быстрый тип набора данных, и он вполне уместен в нашей задаче, так как все, что в ней требуется — это один раз прочитать данные.

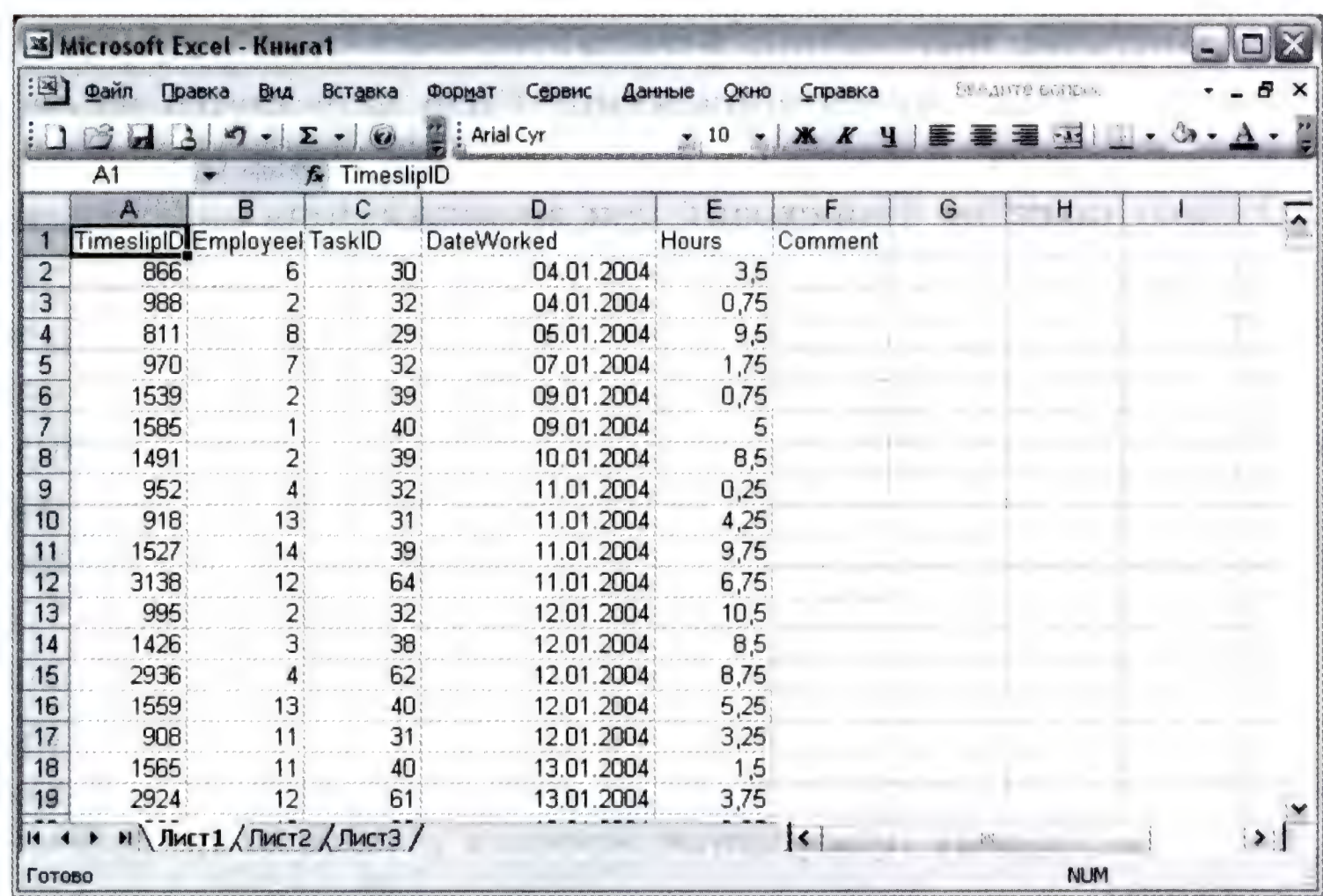
Далее мы запускаем приложение Excel и создаем новый рабочий лист, куда будут помещены данные. Когда приложение Excel запускается из системного меню Пуск, автоматически создается новая рабочая книга с тремя рабочими листами. Однако такое не происходит при автоматизации — приложение открывается пустым. Для создания рабочей книги нужно вызывать метод `Add` коллекции `Workbooks` приложения Excel. После этого свойство `ActiveSheet` объекта `Application` открывает ссылку на рабочий лист по умолчанию.

Процесс копирования данных формально разбит на две части. Вначале мы проходимся в цикле по коллекции `Fields` набора данных и переносим заголовки столбцов в Excel; после этого мы в цикле проходим все записи набора данных и переносим их в ячейки электронной таблицы. Объект `Cells`, дочерний от объекта `Worksheet` приложения Excel, позволяет обращаться к конкретным ячейкам рабочего листа, используя номера строк и столбцов как индексы. При этом приходится выполнять небольшую корректировку, так как столбцы в наборе данных нумеруются начиная с нуля, а на рабочем листе Excel — с единицы.

После создания рабочего листа и перемещения на него данных процедура делает его видимым. Это было сделано в последнюю очередь по той причине, что если отобразить рабочий лист вначале, то процесс переноса данных затормозится, поскольку придется постоянно обновлять экран. Заключительным действием является выход из процедуры. Несмотря на то, что переменная `objXL` при этом выходит за пределы области видимости, серверу автоматизации она уже не нужна. Приложение Excel остается запущенным и отображает данные, показанные на рис. 21.3.

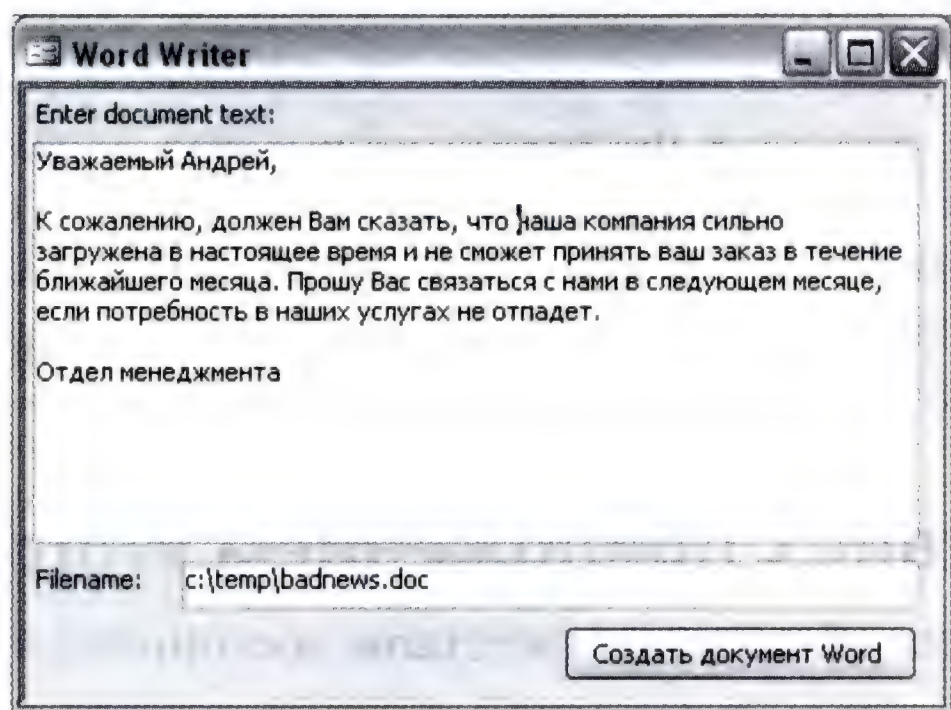
Взаимодействие с приложением Word из Access

Приложение Access обладает множеством достоинств, однако оно не предоставляет хороших средств создания отчетов, содержащих произвольный текст и форматирование. С помощью средств автоматизации это ограничение обойти довольно просто: приложение Microsoft Word идеально подходит для создания профессионального отчета, именно это приложение можно выбрать в качестве сервера автоматизации. На рис. 21.4 показана простая форма Access, позволяющая вводить произвольный текст и имя файла. После щелчка на кнопке создается документ, содержащий введенный текст и сохраненный под именем, введенным в нижнем текстовом поле формы.



	A	B	C	D	E	F	G	H	I
1	TimeslipID	Employee	TaskID	DateWorked	Hours	Comment			
2	866	6	30	04.01.2004	3,5				
3	988	2	32	04.01.2004	0,75				
4	811	8	29	05.01.2004	9,5				
5	970	7	32	07.01.2004	1,75				
6	1539	2	39	09.01.2004	0,75				
7	1585	1	40	09.01.2004	5				
8	1491	2	39	10.01.2004	8,5				
9	952	4	32	11.01.2004	0,25				
10	918	13	31	11.01.2004	4,25				
11	1527	14	39	11.01.2004	9,75				
12	3138	12	64	11.01.2004	6,75				
13	995	2	32	12.01.2004	10,5				
14	1426	3	38	12.01.2004	8,5				
15	2936	4	62	12.01.2004	8,75				
16	1559	13	40	12.01.2004	5,25				
17	908	11	31	12.01.2004	3,25				
18	1565	11	40	13.01.2004	1,5				
19	2924	12	61	13.01.2004	3,75				

Рис. 21.3. Данные Access экспортированы в Excel средствами автоматизации



Word Writer

Enter document text:

Уважаемый Андрей,

К сожалению, должен Вам сказать, что наша компания сильно загружена в настоящее время и не сможет принять ваш заказ в течение ближайшего месяца. Прошу Вас связаться с нами в следующем месяце, если потребность в наших услугах не отпадет.

Отдел менеджмента

Filename: c:\temp\badnews.doc

Создать документ Word

Рис. 21.4. Форма Access для создания документа Word

Текст процедуры обработки щелчка на кнопке довольно прост:

```
Private Sub cmdCreate_Click()  
    ' Копируем текст в документ Word  
    Dim objWord As Word.Application  
  
    ' Иницилируем Word и новый документ  
    Set objWord = New Word.Application  
    objWord.Documents.Add  
  
    ' Вставляем текст после области по умолчанию  
    ' Это - начало документа  
    objWord.ActiveDocument.Range.InsertAfter (txtText.Value)  
  
    ' Сохраняем файл  
    objWord.ActiveDocument.SaveAs (txtFileName.Value)
```



```
' и отображаем его
objWord.Visible = True
End Sub
```

Единственной сложностью в этой задаче была необходимость хорошего знания объектной модели Word, чтобы заставить ее делать то, что нам нужно. Ключевыми действиями были создание документа и добавление в него текста. Для создания документа вызывался метод Add коллекции Documents. Добавление текста — несколько более сложная задача. Документы Word содержат объект Range, который представляет выделенную в настоящий момент область текста. Объект Range имеет массу методов добавления текста, в том числе InsertAfter и InsertBefore. Сразу после создания документа текста в нем еще нет, поэтому объект Range просто указывает на начало документа. Вставляя текст после этой точки, мы вставляем его в самое начало документа — что как раз и нужно. На рис. 21.5 показан результат выполнения процедуры.

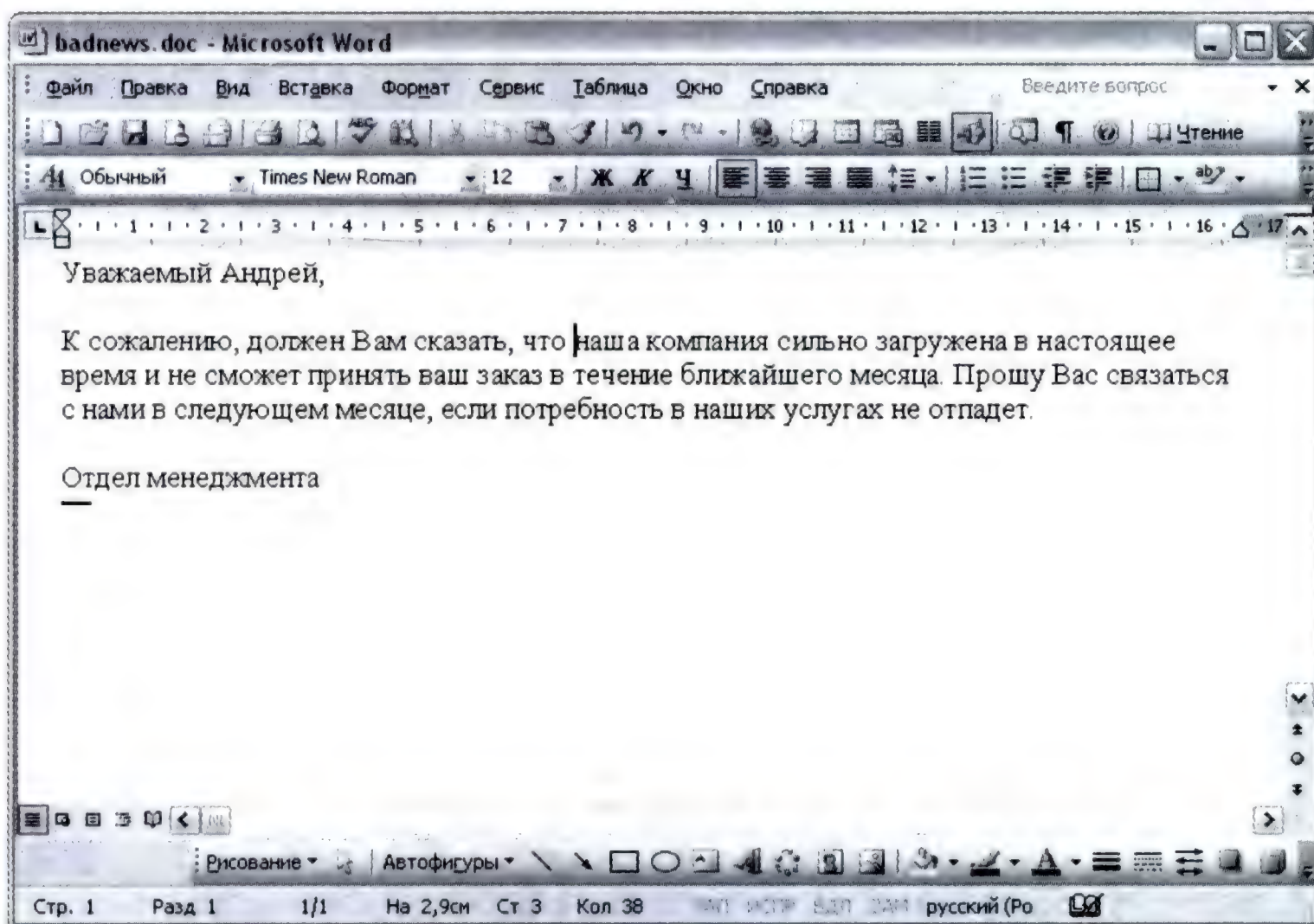


Рис. 21.5. Документ, созданный из Access

ПРАКТИКУМ

Использование диаграмм Excel из Access

Вы уже видели, как автоматизация способна расширить возможности приложений. В этом практикуме на форму Switchboard в базе данных TimeTrack мы поместим новую кнопку. По щелчку на этой кнопке будет открываться приложение Excel, а в нем — создаваться диаграмма, отображающая относительное распре-

ление работы между отдельными сотрудниками компании. Естественно, в этой процедуре сервером автоматизации будет Excel.

```

Sub ChartInExcel()
' Отправка результатов запроса WeeklyTimeslips
' в Excel для дальнейшего анализа

Dim rstHoursByEmployee As ADODB.Recordset
Dim objXL As Excel.Application
Dim objWS As Excel.Worksheet
Dim objChart As Excel.Chart

Dim fld As ADODB.Field
Dim intCol As Integer
Dim intRow As Integer

Set rstHoursByEmployee = New ADODB.Recordset

' Помещаем необходимые данные в набор записей
rstHoursByEmployee.Open _
    "HoursByEmployee", CurrentProject.Connection

' Запускаем Excel
Set objXL = New Excel.Application
' Создаем новый рабочий лист
objXL.Workbooks.Add
Set objWS = objXL.ActiveSheet

' Копируем данные
' Вначале копируем имена полей
For intCol = 0 To rstHoursByEmployee.Fields.Count - 1
    Set fld = rstHoursByEmployee.Fields(intCol)
    objWS.Cells(1, intCol + 1) = fld.Name
Next intCol
' а затем - фактические данные
intRow = 2
Do Until rstHoursByEmployee.EOF
    For intCol = 0 To rstHoursByEmployee.Fields.Count - 1
        objWS.Cells(intRow, intCol + 1) = _
            rstHoursByEmployee.Fields(intCol).Value
    Next intCol
    rstHoursByEmployee.MoveNext
    intRow = intRow + 1
Loop

' Добавляем новую диаграмму
objXL.Charts.Add
Set objChart = objXL.ActiveChart

' Настраиваем диаграмму
objChart.ChartType = xl3DPieExploded
objChart.SetSourceData _
    source:=objWS.Range("A1:B" & CStr(intRow - 1)), _
    PlotBy:=xlColumns
objChart.Location Where:=xlLocationAsNewSheet
objChart.HasTitle = True
objChart.ChartTitle.Characters.Text = "Распределение работы"

```



```
' Делаем рабочий лист видимым
objXL.Visible = True

' Метод Excel Quit мы не вызываем, поскольку
' рабочий лист должен быть доступным
' после завершения процедуры
End Sub
```

С большей частью этой программы вы уже знакомы по материалу предыдущего раздела: мы открываем набор данных на запросе HoursByEmployee и передаем данные в Excel. После этого создается диаграмма. Это — пример еще одного использования объектной модели Excel. Вначале мы вызываем метод Add коллекции Charts и создаем новую диаграмму. Указатель на активную диаграмму мы сохраняем в переменной objChart для дальнейшего использования.

Объект Chart имеет множество свойств и методов, но в данном случае нужно установить только малую их часть. В примере мы выбираем диаграмму типа трехмерного пирога, в которую попадают данные из записанной области ячеек рабочего листа. Эту диаграмму мы сохраняем как новый рабочий лист с заданным именем.

На рис. 21.6 показан результат запуска этой процедуры. Естественно, эта диаграмма далека от идеала, но в ней использовано достаточное количество цветов, чтобы представить множество сотрудников.

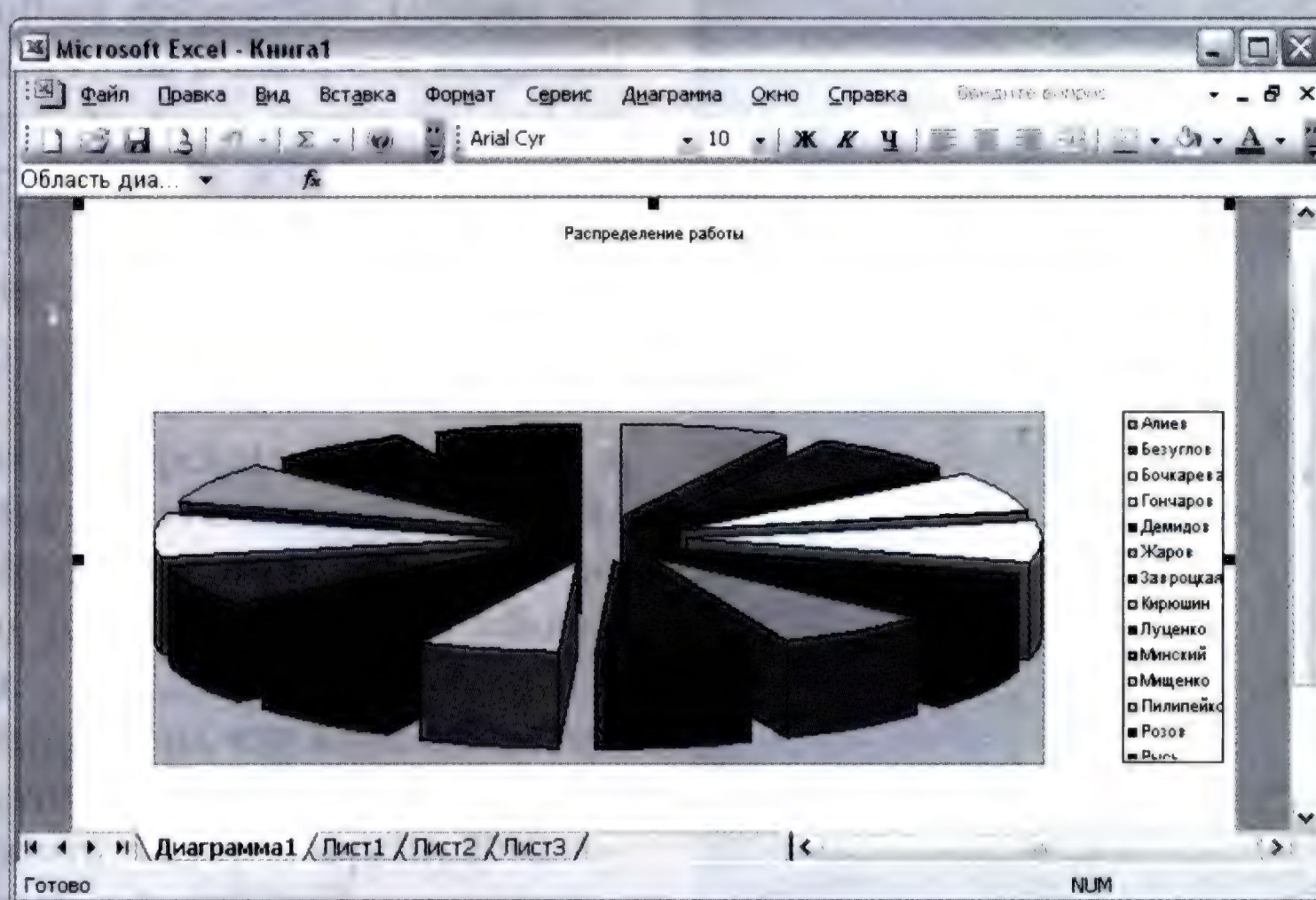


Рис. 21.6. Диаграмма Excel, созданная из Access

Работа с файлами XML

22

Введение в XML

Возможно вы сталкивались с проблемой, когда невозможно было совершить обмен данными с Access лишь по той причине, что данный формат это приложение не понимало. Именно здесь на помощь может прийти язык *XML* (Extensible Markup Language). Технические специалисты определяют XML как платформо-независимый язык разметок, созданный для структурированных документов. Однако поскольку эта глава посвящена вопросам автоматизации всего процесса, мы не будем углубляться в основы этого языка, следует лишь учесть, что XML-технология позволяет обмениваться данными независимо от их исходного формата. Если некоторое приложение может экспортировать свои данные в файл XML, Access сможет их импортировать. Язык XML при этом берет на себя всю нагрузку в задачах обмена данными.

Несмотря на то, что XML существует уже долгое время, Access начал его поддерживать только в версии 2002 года. Программа Access 2003 поддерживает следующие типы файлов XML:

- **XML.** Стандарт форматирования для структурированных файлов. Файл XML для организации данных в блоки использует дескрипторы.

В ЭТОЙ ГЛАВЕ

Введение в XML	373
Использование метода ExportXML	374
Использование метода ImportXML	379

- **XSL.** Стандарт для преобразования импортированных или экспортированных данных. Файл XSL используют для форматирования данных в файле XML. Часто эти два файла объединяют для внедрения данных XSL в файл XML.
- **XSD.** Стандарт, определяющий или описывающий структуру базы данных в терминах XML. Эти файлы *определения схемы* содержат типы данных, а также правила данных и дескрипторов соответствующего файла XML.

Изучив материал этой главы, вы научитесь работать со всеми этими тремя типами файлов. Вы узнаете, как XML используется для хранения данных, XSL — для форматирования, а XSD — для хранения информации о структуре данных.

Примечание

Структурированным называют такой тип текстового файла, в котором для идентификации данных и метаданных использованы *дескрипторы*. К примеру, файл XML, содержащий данные о заказчиках, может содержать дескрипторы <CustomerName>, <CustomerAddress> и другие. Access может интерпретировать эти дескрипторы как заголовки столбцов. Скорее всего, фактические данные этих столбцов будут обрамлены соответствующими дескрипторами.

В дополнение следует сказать, что средства поддержки XML в Access позволяют создавать документы HTML, готовые для помещения в Web. Более подробно об этом вы узнаете в разделе “Экспорт файлов для Web” далее в этой главе.

Использование метода ExportXML

Данные Access можно напрямую экспортировать в файл XML, вызвав метод ExportXML языка VBA. Он имеет следующий синтаксис:

```
приложение.ExportXML тип_объекта, источник_данных _  
    [, файл_данных] [, файл_схемы] [, файл_представления] _  
    [, файл_рисунка] [, кодировка] [, флаги] _  
    [, критерий_фильтра] [, дополнительные_данные]
```

где *приложение* представляет модель Access. В табл. 22.1 перечислены аргументы этой функции. Несмотря на то, что аргументы *цель_данных*, *цель_схемы* и *цель_представления* не являются обязательными, из них вы должны предоставить по крайней мере один. В табл. 22.2 и 22.3 перечислены константы, используемые отдельными аргументами метода ExportXML.

Таблица 22.1. Аргументы метода ExportXML

Аргумент	Описание
<i>тип_объекта</i>	Внутренняя константа, определяющая тип экспортируемого объекта (см. табл. 22.2)
<i>источник_данных</i>	Строковое значение, идентифицирующее объект, экспортируемый по имени
<i>файл_данных</i>	Путь и имя файла XML, в который экспортируются данные
<i>файл_схемы</i>	Путь и имя файла XSD, в который экспортируется схема данных
<i>файл_представления</i>	Путь и имя файла XSL, в который экспортируется информация о представлении
<i>файл_рисунка</i>	Путь для экспортируемой графики
<i>кодировка</i>	Одна или две константы (acUTF16 и acUTF8), определяющие кодировку
<i>флаги</i>	Значение битовой маски (см. табл. 22.3). Складывая отдельные значения, вы можете получить их комбинацию
<i>критерий_фильтра</i>	Строковое значение, ограничивающее экспортируемые данные путем применения критерия отбора
<i>дополнительные_данные</i>	Определяет дополнительные таблицы для экспорта. Игнорируется, если в аргументе <i>флаги</i> установлено значение acLiveReportSource

Таблица 22.2. Константы типа объекта

Объект	Константа
Form (Форма)	acExportForm
Function (Функция)	acExportFunction
Query (Запрос)	acExportQuery
Report (Отчет)	acExportReport
View (Представление)	acExportView
Stored Procedure (Хранимая процедура)	acExportStoredProcedure
Table (Таблица)	acExportTable

Таблица 22.3. Аргументы метода ExportXML

Значение	Константа	Описание
1	acEmbedSchema	Внедрить информацию о схеме в документ XML (определенный в аргументе <i>файл_данных</i>)
2	acExcludePrimaryKey- AndIndexes	Исключить из экспортируемой схемы первичные ключи и индексы
4	acRunFromServer	Создать для экспортируемого отчета оболочку активных серверных страниц ASP
8	acLiveReportSource	Создать для экспортируемого отчета ссылку на удаленную базу данных Server 2000
16	acPresistReportML	Задать для экспортируемого объекта тип ReportML — тип XML, специально разработанный для отчетов Access

Предупреждение

Функция ExportXML переписывает уже существующие файлы. Для того чтобы избежать потери информации на диске, перед началом процесса экспорта проверьте наличие одноименных с экспортируемым файлов. В случае обнаружения таковых переименуйте их, задайте новое имя для экспортируемого файла или просто завершите задачу.

Пример экспорта

В простейшем случае экспорта в XML создается один файл XML. Приведенная в качестве примера процедура выполняет экспорт таблицы клиентов в файл XML.

```
Sub ExportTable(source As String, path As String, _  
    target As String)  
    'Экспорт источника данных в файл XML  
    Access.Application.ExportXML acExportTable, source, target  
End Sub
```

Введите эту процедуру в стандартный модуль или используйте модуль Chapter22 базы данных примеров. При вызове процедуры передайте ей имя экспортируемой таблицы, а также полный путь и имя файла, в которые следует экспортировать данные. К примеру, в окне Immediate вы можете ввести следующую инструкцию и экспортировать данные о клиентах:

```
ExportTable "Clients", путь, "Clients.xml"
```

(Убедитесь, что аргумент *путь* соответствует структуре файлов вашего компьютера.) Обязательно включите расширение `.xml` в последний аргумент; в противном случае экспортированный файл не будет распознан редакторами и прочими утилитами как файл XML. (Учтите, что аргумент *путь* является

строкой, так что не забудьте заключить его в кавычки.) На рис. 22.1 показан экспортированный файл XML, открытый в приложении Internet Explorer.

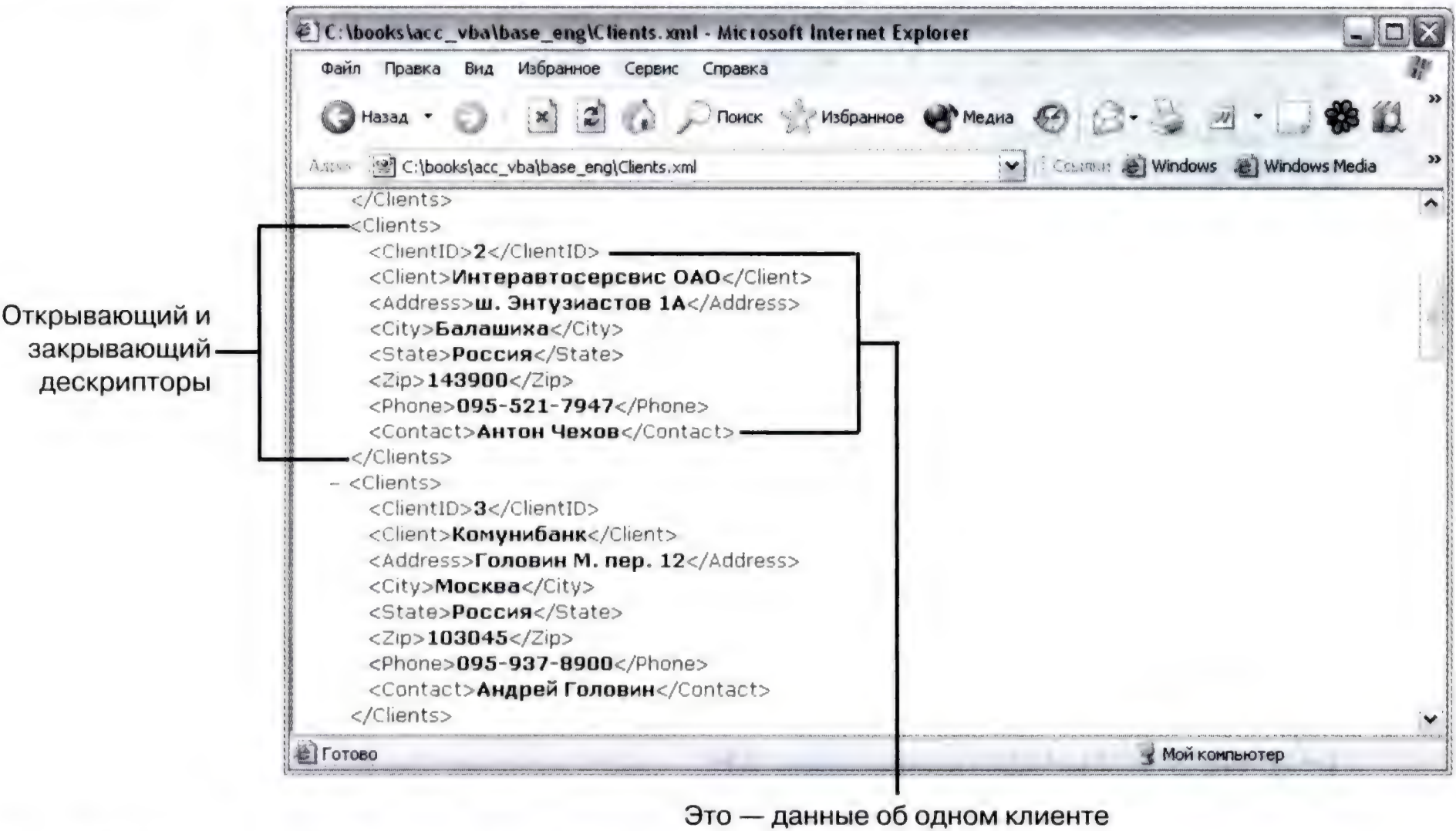


Рис. 22.1. Этот файл XML содержит дескрипторы и фактические данные, однако его трудно осмыслить с использованием стандартных средств

Формат XML в своем естественном виде практически невозможно воспринять обычному пользователю. Возможно, вы никогда не просматривали файлы этого типа, если только вам не приходилось редактировать их содержание.

Приведенная выше процедура работает только с таблицами. Вы можете в нее добавить еще один аргумент и дать возможность пользователю указывать дополнительно тип экспортируемого объекта.

Экспорт файлов для Web

Для того чтобы быстро создать документ, готовый для помещения в Web, измените расширение аргумента *файл_данных* на *.html*. Результат не будет безупречным, однако на этот раз браузер будет отображать только данные, без дескрипторов.

Выполните следующую инструкцию в окне Immediate (она полностью совпадает с предыдущей за исключением того, что расширение *.xml* заменено на *.html*):

```
ExportTable "Clients", путь, "Clients.html"
```

Теперь откройте файл *Clients.html* в своем браузере (используйте Проводник для поиска файла в текущей директории, после чего дважды щелкните на нем). На рис. 22.2 показан этот файл, открытый в приложении Internet Explorer. Он не совершенен, однако мы достаточно быстро и легко получили такой результат.

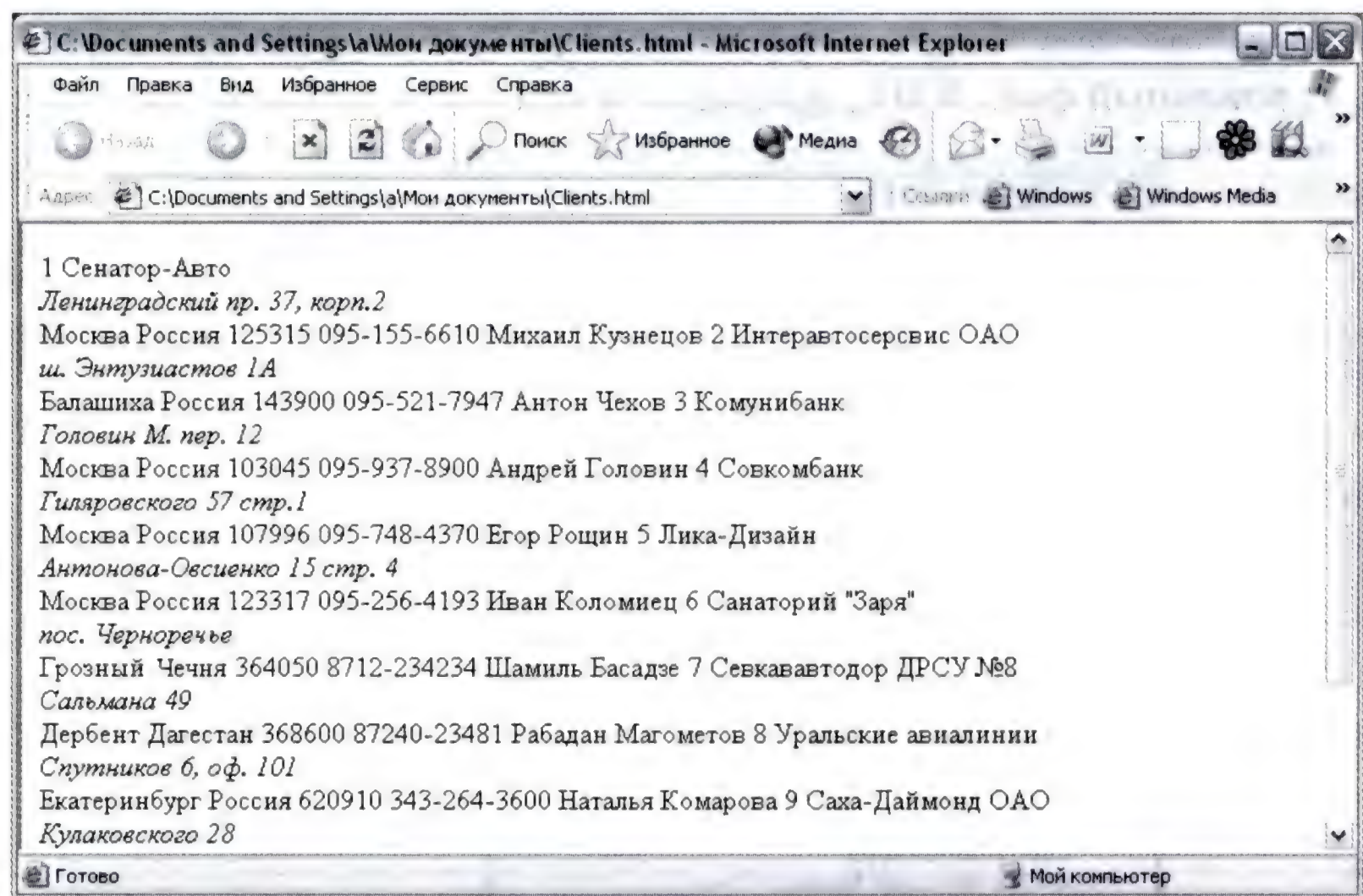


Рис. 22.2. Изменив расширение при вызове процедуры, мы быстро получили результат, готовый для помещения в Web

Экспорт связанных данных

Экспорт связанных данных достаточно прост, если использовать интерфейс пользователя Access. Однако автоматизация этого процесса вызывает определенные сложности. Аргумент *дополнительные_данные* метода `ExportXML` позволяет определять дополнительные таблицы для экспорта, но просто перечислить их имена в списке недостаточно — для добавления таблиц придется воспользоваться специальным объектом `AdditionalData`.

Для экспорта связанных данных необходимо идентифицировать связанные таблицы как объект `AdditionalData`, а после этого передать ссылку на него аргументу *дополнительные_данные*. В предлагаемой процедуре такой объект создается, после чего используется его метод `Add` для добавления в коллекцию таблиц.

```
Sub ExportMultipleTables(source As String, path As String, _
    target As String, addtable As String)
    'Экспорт связанных таблиц в файл XML
    Dim objAddTable As AdditionalData
    Set objAddTable = Application.CreateAdditionalData
    objAddTable.Add (addtable)
    Access.Application.ExportXML acExportTable, source, target, _
        , , , , , objAddTable
End Sub
```

Для демонстрации этой процедуры выполните следующую инструкцию в окне `Immediate` (только не забудьте подставить *путь*):

```
ExportMultipleTables "Clients", путь, "ClientsAndProjects.html", _
    "Projects"
```


На рис. 22.3 показаны все клиенты с открытыми для них проектами. Несмотря на то, что формату не хватает достаточной структурированности, здесь представлены все данные.

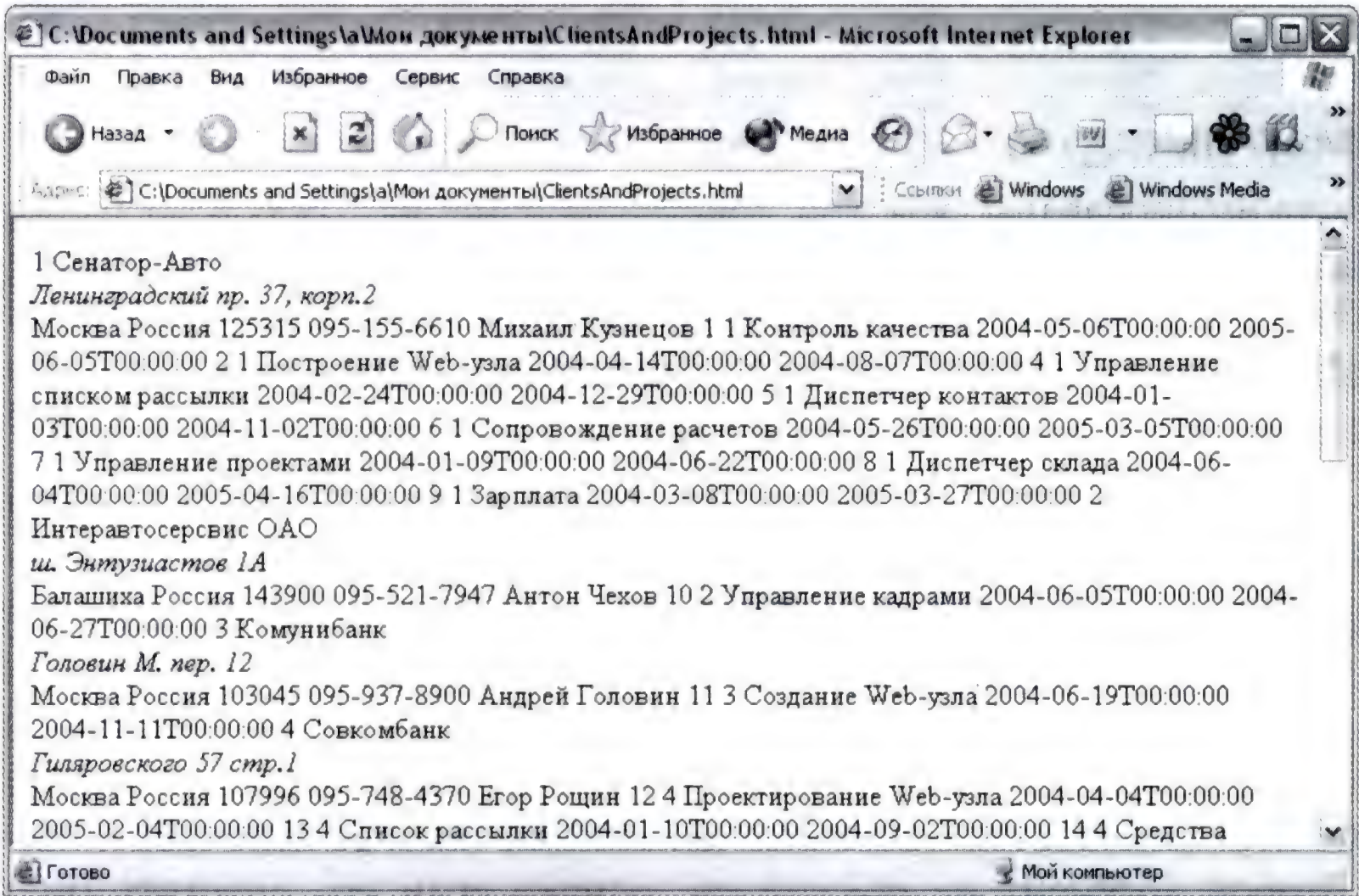


Рис. 22.3. Программный экспорт связанных данных в файл XML

Использование метода ImportXML

Нужные данные не всегда хранятся только в одной базе данных — иногда приходится импортировать и внешние данные. Для этого предназначен Мастер импорта программы Access, который поддерживает массу разных форматов. Однако для импорта данных XML не обязательно использовать этот мастер. Вы можете воспользоваться методом ImportXML, позволяющим импортировать данные из файлов XML, который имеет следующий синтаксис:

`приложение.ImportXML источник_данных[, параметр]`

где *источник_данных* — обязательный аргумент строкового типа, идентифицирующий полный путь и имя импортируемого файла XML, а *параметр* — одна из встроенных констант, перечисленных в табл. 22.4.

Таблица 22.4. Константы для метода ImportXML

Константа	Значение	Описание
acStructureOnly	0	Импортировать только структуру таблицы
acStructureAnd-Data	1	Импортировать структуру таблицы и ее данные (используется по умолчанию)
acAppendData	2	Добавление данных в существующую таблицу

Импорт данных XML не так сложен, как экспорт. Обычно нужно определить сам файл, который Access по умолчанию импортирует в новую таблицу. Иногда может потребоваться импортировать только структуру или добавить данные в существующую таблицу — тогда воспользуйтесь константами, перечисленными в табл. 22.4.

Пример импорта

А теперь проследим, как три константы изменяют результат импорта одного и того же файла данных о клиентах, экспортированного в предыдущем разделе. Для этого введите следующую процедуру в стандартный модуль или воспользуйтесь готовой в модуле Chapter22 базы данных примеров.

```
Sub ImportXMLData(source As String, Optional impopt As Variant)
    'импорт файла XML
    'при необходимости с путем
    If IsMissing(impopt) = True Then
        impopt = acStructureAndData
    End If
    Access.Application.ImportXML source, impopt
End Sub
```

Эта процедура принимает имя импортируемого файла и константу. Вы можете передать как текстовое имя константы, так и ее целочисленное значение. Второй аргумент не обязателен, однако если его опустить, процедура предположит, что вы хотите использовать тип импорта по умолчанию — структуры и данных.

Перед тем как что-либо импортировать, закройте все окна приложения Word и браузера, в которых открыты файлы .xml и .html из предыдущих примеров. В некоторых случаях открытый файл .xml может вызвать сообщение об ошибке.

В окне Immediate выполните следующую инструкцию:

```
ImportXMLData "путь\clients.xml"
```

(Если вы сохранили файл экспорта Clients.xml не в текущей директории, обязательно включите в аргумент *путь*.) После запуска этой инструкции перейдите в окно База данных Access и проверьте наличие новой таблицы с названием Clients1. На рис. 22.4 показана эта новая таблица в табличном представлении — она идентична таблице Clients. Единственное отличие состоит в том, что новая таблица не имеет ни ключей, ни индексов. Во время процесса импорта к имени таблицы Access добавил единицу, так как с этим именем (Clients) в базе данных таблица уже существует.

ClientID	Client	Address	City	State	Zip	Phone	Contact
1	Сенатор-Авто	Ленинградский	Москва	Чукотка	125315	095-155-6610	Михаил Кузнецов
2	Интеравтосерс	ш. Энтузиастов	Балашиха	Россия	143900	095-521-7947	Антон Чехов
3	Комунибанк	Головин М. пер.	Москва	Россия	103045	095-937-8900	Андрей Головин
4	Совкомбанк	Гиляровского 5	Москва	Россия	107996	095-748-4370	Егор Рошин
5	Лика-Дизайн	Антонова-Овсина	Москва	Россия	123317	095-256-4193	Иван Коломиец
6	Санаторий "Заря"	пос. Черноречье	Грозный	Чечня	364050	8712-234234	Шамиль Басаев
7	Севкававтодор	Сальмана 49	Дербент	Дагестан	368600	87240-23481	Рабдан Магомедов
8	Уральские авиалинии	Спутников 6, офис	Екатеринбург	Россия	620910	343-264-3600	Наталья Комарова
9	Саха-Даймонд	Кулаковского 2	Якутск	Саха	677007	4112-260501	Павел Сулейманов
10	паб O'Brien's	Михайловская	Киев	Украина	01001	044-2472279	Михаил Задорин
11	Астро-Волга	О. 8 Марта 66, оф.	Екатеринбург	Россия	620063	343-251-7607	Илья Суровцев

Рис. 22.4. С помощью метода ImportXML можно легко импортировать данные XML в новую таблицу

В данном случае нельзя импортировать файл Clients.xml с использованием константы acAppendData, так как это приведет к дублированию уже существующих записей таблицы и будет противоречить существующим правилам (индексам). При попытке такого импорта откроется сообщение об ошибке, показанное на рис. 22.5. Access импортирует только те записи, которые не нарушают ключ. Если некоторая запись вызывает проблемы и возвращает ошибку, Access протоколирует информацию о ней в таблице с именем ImportErrors.

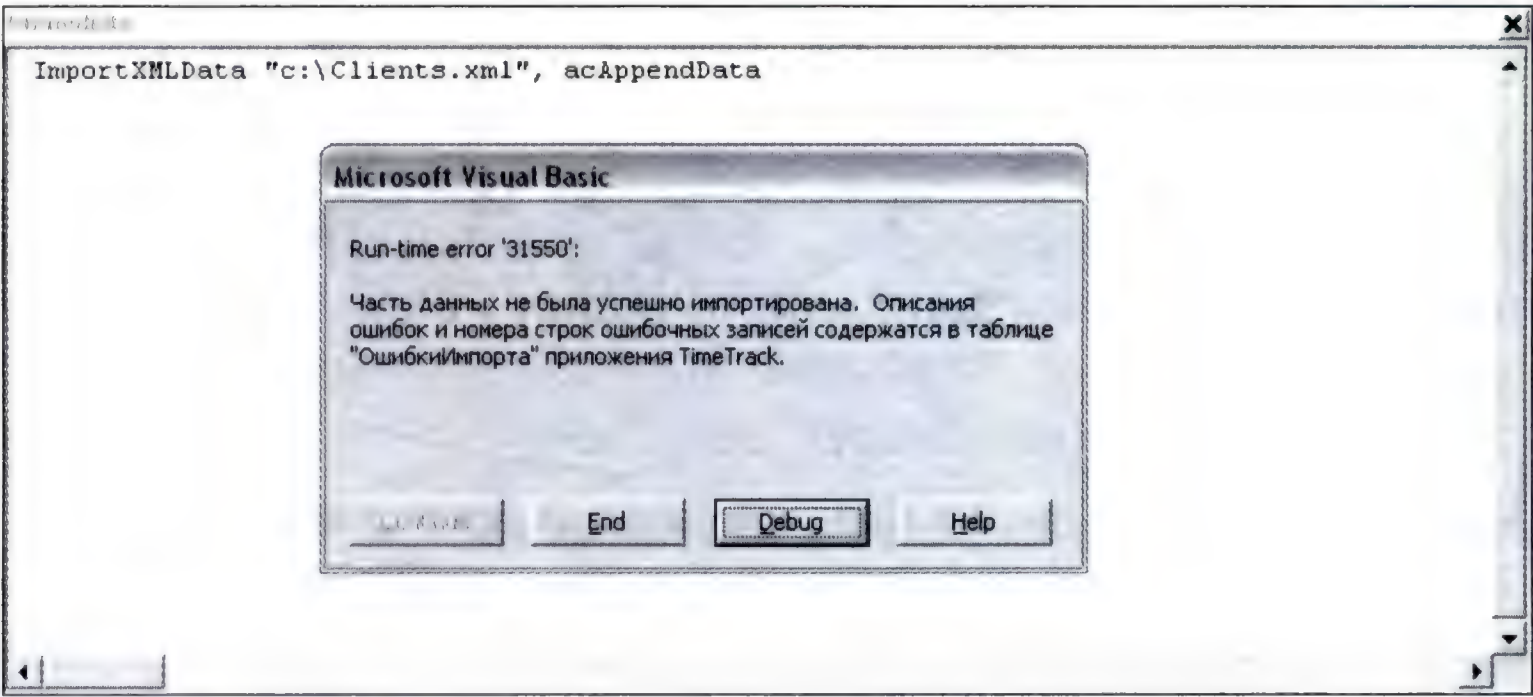


Рис. 22.5. Если при импорте данных нарушаются какие-либо правила базы данных, Access возвращает ошибку и не импортирует такие данные

В данном случае проблема состоит в том, что первичный ключ ClientID не может содержать двух одинаковых значений. Если открыть любыми средствами (с помощью Word или любого другого редактора XML) файл clients.xml и исправить в нем значение ClientID в первой записи с единицы, к примеру, на сто, эта запись будет импортирована несмотря на то, что все остальные поля будут полностью дублировать первую запись таблицы Clients. Для импорта только структуры таблицы выполните в окне Immediate следующую инструкцию:


```
ImportXMLData "путь\clients.xml", acStructureOnly
```

Обязательно включите в первый аргумент полный путь к файлу и расширение .xml. На рис. 22.6 показан результат выполнения этой инструкции — новая таблица базы данных. Обратите внимание, что на этот раз таблице присвоено имя Clients2. Как и раньше, Access увеличивает счетчик в конце исходного имени на единицу, поскольку таблицы Clients и Clients1 уже существуют.

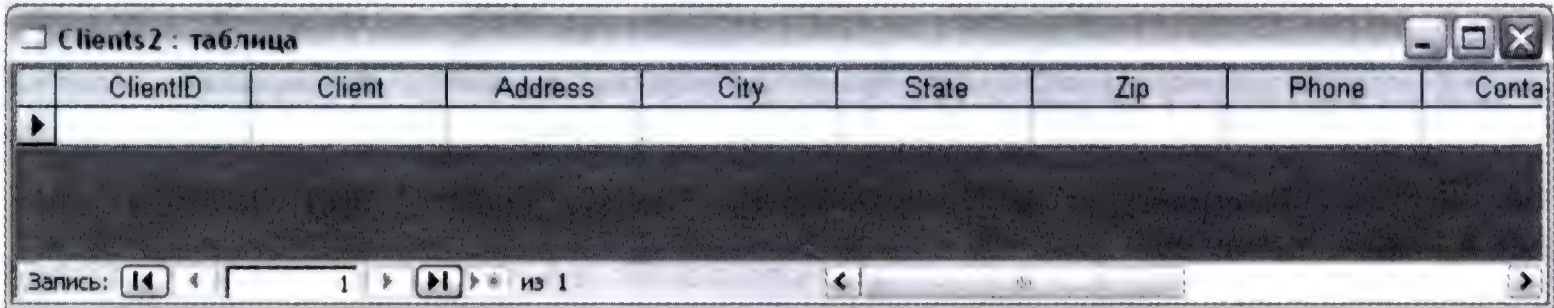


Рис. 22.6. При желании вы можете импортировать только структуру таблицы

Предупреждение

Импорт структуры файла XML вызывает создание пустой таблицы с теми же полями и свойствами. В то же время этот процесс не копирует исходные первичные ключи и индексы.

В предыдущем разделе, посвященном экспорту, вы узнали, как можно импортировать в файл XML связанные таблицы с помощью объекта AdditionalData. Импорт связанных таблиц происходит гораздо проще, поскольку Access управляет процессом самостоятельно. В файле XML указано, что таблицы являются связанными, а значит импортируются, не требуя от вас никакой дополнительной информации. Чтобы проиллюстрировать, как Access справляется со связанными данными, выполним в окне Immediate следующую инструкцию:

```
ImportXMLData "путь\ClientsAndProjects.html"
```

(Не забудьте заменить *путь* на фактическую директорию). Результат показан на рис. 22.7 — это таблицы Clients3 и Project1. Обратите внимание, что в примере импортировался файл .html, а не .xml. Метод ImportXML достаточно гибок для того, чтобы обработать данные формата HTML; несмотря на это все же желательно с его помощью импортировать файлы XML.

Предупреждение

Access может импортировать связанные данные из файла XML и разделить их по таблицам, однако он не может воссоздать отношения между этими таблицами — их придется создавать вручную.

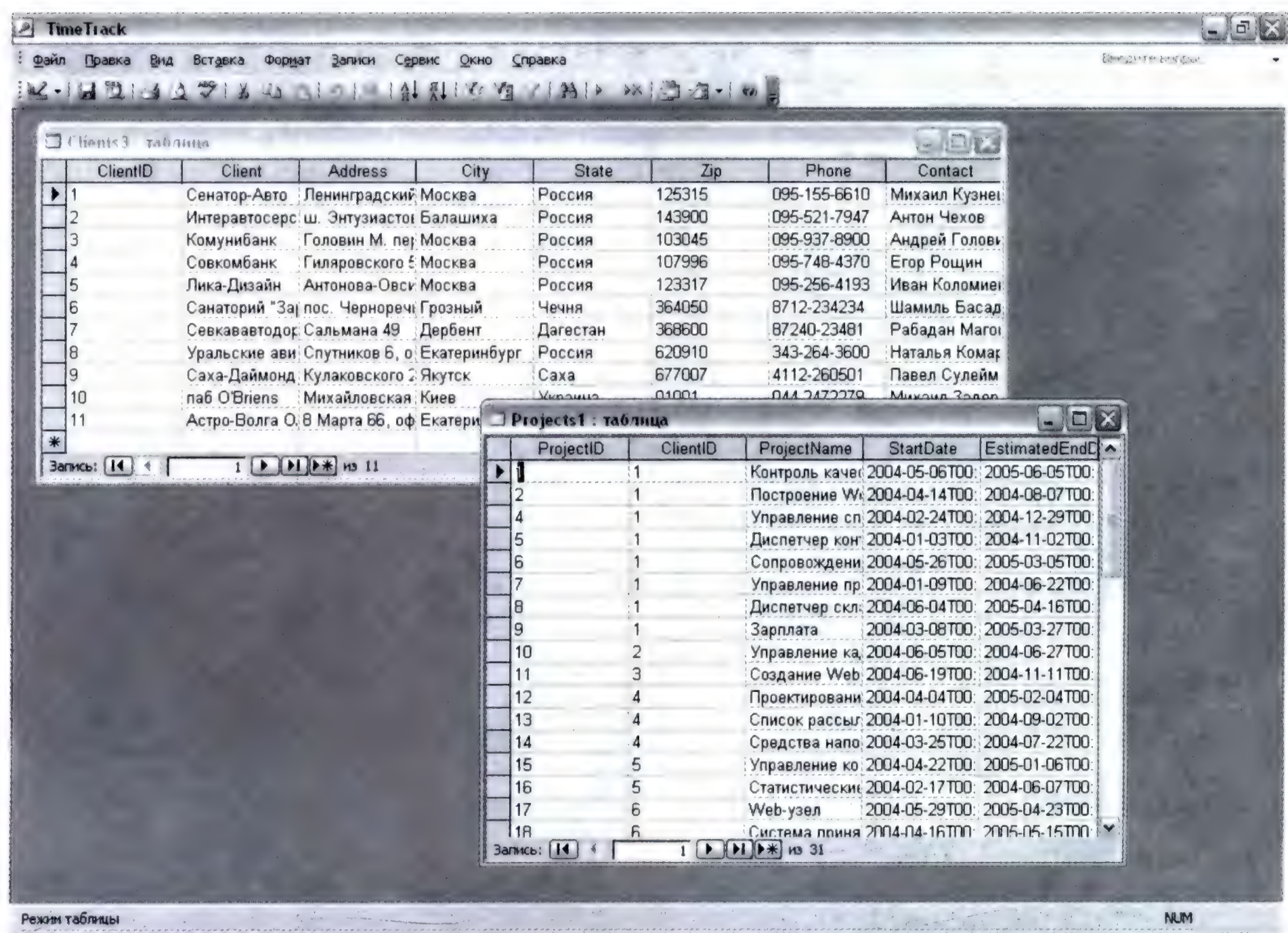


Рис. 22.7. Метод ImportXML с легкостью обрабатывает связанные данные

ПРАКТИКУМ

Экспорт текущей информации о проектах

Предположим, что разработчику часто приходится тратить время на посещение офиса клиента, чтобы работать с текущими данными проектов. Чтобы облегчить ему эту задачу, можно экспортировать интересующие его данные в файл XML или HTML и переслать ему их средствами электронной почты прямо на его постоянное рабочее место. При желании вы можете автоматизировать процесс отправки файла по электронной почте или даже предоставить доступ разработчику непосредственно к базе данных через Internet (в предлагаемом примере мы не будем реализовывать эти две возможности).

Для начала создадим новую пустую форму с элементом списка и двумя командными кнопками. В табл. 22.5 перечислены значения свойств этой формы и элементов управления.

Таблица 22.5. Свойства формы и элементов управления

Объект	Свойство	Значение
форма	Имя	Export
	Подпись	Экспорт проектов клиента
	Полосы прокрутки	Отсутствуют
	Область выделения	Нет
	Кнопки перехода	Нет
список	Имя	lstClients
	Источник строк	SELECT Clients.ClientID, Clients.Client FROM Clients
	Число столбцов	2
	Ширина столбца	0 см
кнопка	Имя	cmdCancel
	Подпись	Отмена
кнопка	Имя	cmdExport
	Подпись	Экспорт

В модуль формы введите следующие процедуры:

```
Private Sub cmdCancel_Click()  
    'очистить выбор  
    lstClients.Value = ""  
End Sub  
  
Private Sub cmdExport_Click()  
    'Экспорт проектов выделенного клиента в файл  
    'HTML в текущей директории  
    Dim objProjects As AdditionalData  
    On Error GoTo HandleErr  
    Set objProjects = Application.CreateAdditionalData  
    objProjects.Add ("Projects")  
    Application.ExportXML acExportTable,  
        "Clients", "ClientsAndProjects.HTML",  
        , , , , "ClientID = " & lstClients, objProjects  
    Exit Sub  
  
ExitHere:  
    Set objProjects = Nothing  
    Exit Sub  
HandleErr:  
    MsgBox "Ошибка " & Err.Number & ": " & _  
        Err.Description  
    Resume ExitHere  
End Sub
```

Сохраните форму и откройте ее в представлении формы (рис. 22.8).

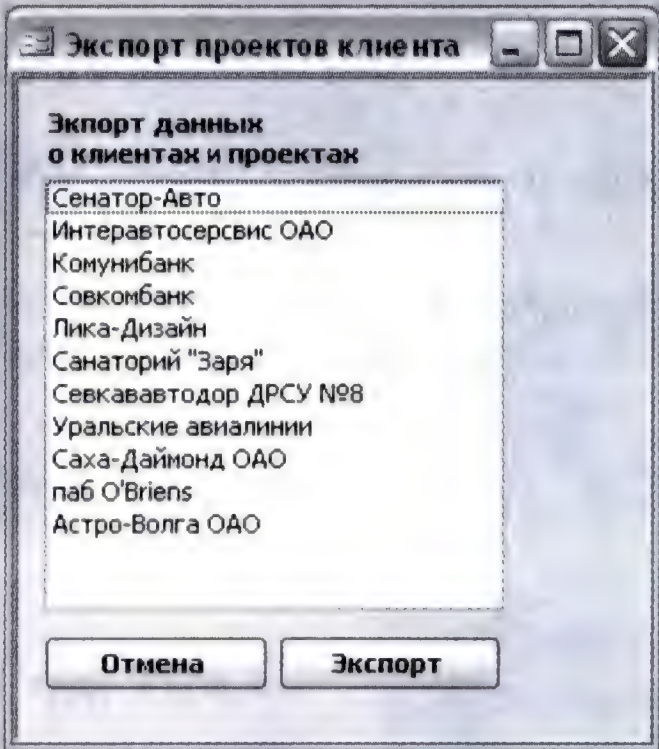


Рис. 22.8. Для выбора клиента используйте эту простую форму

В списке выберите пункт Сенатор-Авто ОАО. В настоящее время процедура может за один раз обработать только одного клиента. Однако в форме был использован элемент простого списка, так что можно расширить используемые средства на множество выбранных клиентов.

После выбора клиента щелкните на кнопке Экспорт. Видимого эффекта не произойдет, однако в текущую директорию в файл HTML будут экспортированы данные о связанных с выбранным клиентом проектах. Щелкните в этой директории на файле ClientsAndProjects.html.

Теперь можно отправить по электронной почте файл экспорта разработчику. Однако сейчас мы этого делать не будем, а просто откроем файл в текущем браузере (рис. 22.9). В этом файле содержатся только проекты клиента, выбранного нами в форме.

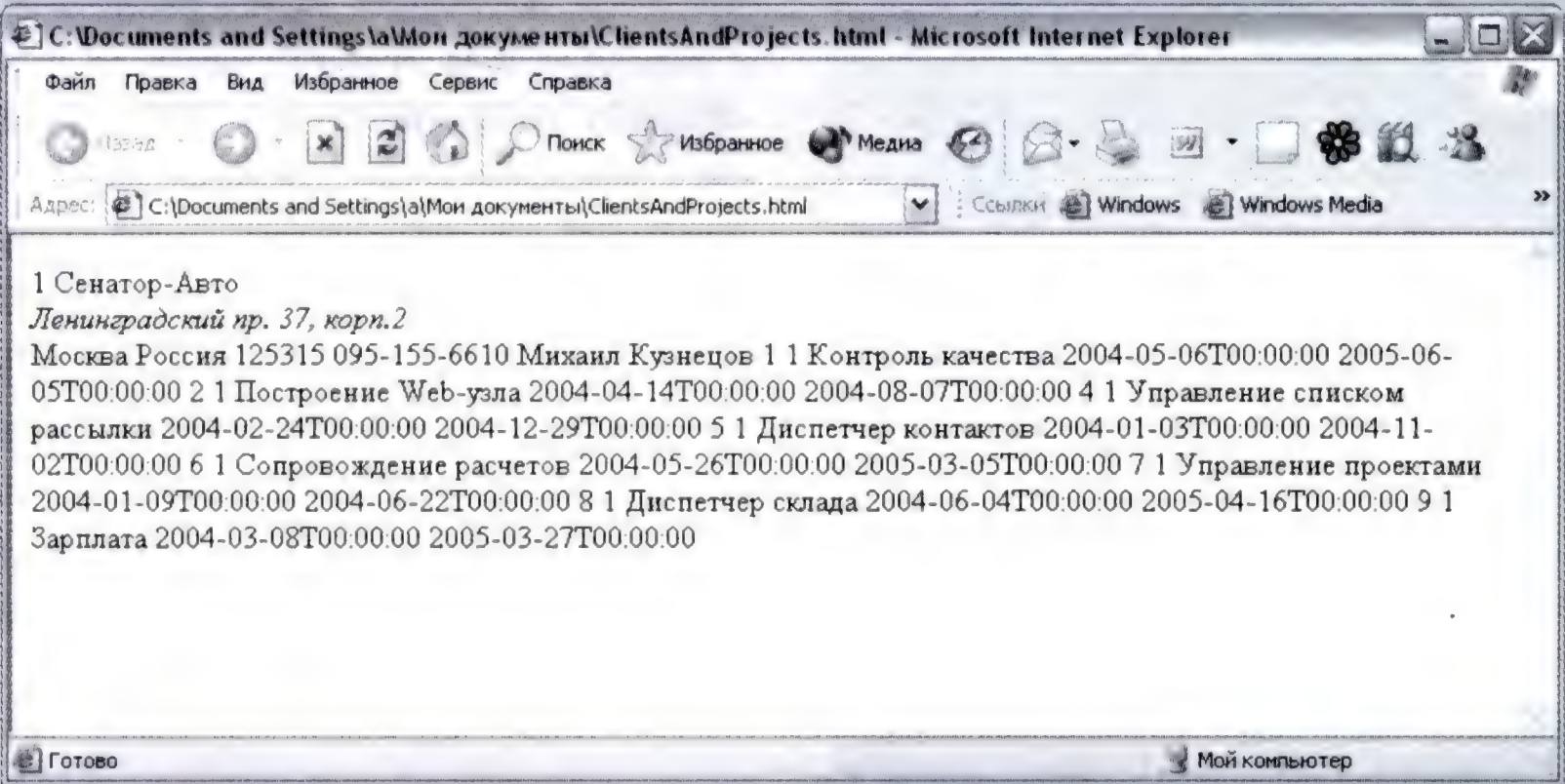


Рис. 22.9. Откройте файл экспорта в браузере и просмотрите проекты клиента

Предупреждение

Если в текущей директории уже существует файл с таким же именем, он будет переписан. Если вы хотите отслеживать этот вопрос, добавьте в процедуру фрагмент программы, проверяющий существование одноименного файла перед экспортом. При этом вы сможете как отменить экспорт, так и изменить имя любого из файлов.

Использование Windows API

23

Объявление вызовов API

Вы, наверняка, знаете, что VBA — не единственный язык программирования приложений под Windows. Существуют сотни альтернатив, таких как C#, C++, Visual Basic .NET и другие. Вы не можете использовать языки, отличные от VBA, для программирования в Access тем же способом, что и VBA, однако для программирования в среде Windows они доступны.

Несмотря на то, что разные языки программирования предлагают собственные способы решения одних и тех же задач, они обладают одной ключевой особенностью: все они работают в операционной среде Windows. Взаимодействие с этой средой реализуется с помощью слоя программного обеспечения, называемого *интерфейсом программирования приложений Windows*, или *Windows API*. Интерфейс Windows API, созданный компанией Microsoft, содержит тысячи функций, предназначенных для выполнения таких задач, как рисование окон, передача информации по сетям и т.д.

В ЭТОЙ ГЛАВЕ

Объявление вызовов API.....	387
Использование вызовов API	389
Вызовы API, которые можно использовать в Access.....	390
Когда использовать Windows API.....	393

Порой отдельные составляющие Windows API недоступны непосредственно в языке программирования. К примеру, VBA не позволяет напрямую получить имя компьютера, на котором запущена программа несмотря на то, что эта функция является частью Windows API. Однако язык VBA содержит в себе нечто другое — механизм общего назначения, позволяющий вызывать процедуры интерфейса Windows API, что обычно называют *вызовами API*. В настоящей главе будут раскрыты основы этого механизма.

Использование вызовов Windows API представляет собой двухступенчатый процесс. В первую очередь нужно *объявить* вызов API, который вы собираетесь использовать. При этом устанавливается подключение VBA к соответствующим программам Windows API. После этого можно вызывать нужную процедуру данного интерфейса так же, как и любую другую.

Примечание

Прямой вызов Windows API — более сложный вопрос. Ознакомившись с материалом настоящей главы, вы сможете представить себе объем тех возможностей, которые открываются в случае прямого вызова Windows API.

В следующем примере мы используем вызов функции Windows API с названием `GetComputerName` для получения имени локального компьютера. Вот как можно его объявить в разделе общих объявлений модуля VBA:

```
Declare Function GetComputerName _
    Lib "kernel32" Alias "GetComputerNameA" _
    (ByVal lpBuffer As String, nSize As Long) _
    As Long
```

Это объявление сообщает интерпретатору VBA:

- что именем функции в коде VBA будет `GetComputerName`;
- что фактическая функция содержится в библиотеке Windows с именем `kernel32`;
- имя фактической функции в этой библиотеке — `GetComputerNameA`;
- функция принимает два аргумента: один из них имеет тип `String`, другой — `Long`;
- функция возвращает значение типа `Long`.

Совет

Файлы справки VBA не содержат никакой информации о вызовах Windows API. Вместо справки вам придется обращаться к справочнику *Platform SDK* (Software Developers Kit), находящемуся в Internet. К примеру, документацию к функции `GetComputerName` можно найти по адресу <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/getcomputername.asp>.

Использование вызовов API

Поскольку вы уже знакомы с программированием на языке VBA, вы должны знать, что не существует никаких различий между процедурами, объявленными в VBA с помощью ключевых слов `Sub` и `Function`, и теми, которые объявлены с помощью ключевого слова `Declare`. Вот пример функции VBA, которая извлекает имя компьютера с помощью вызова API, объявленного ранее:

```
Function GetThisComputerName() As String
    ' Получение имени локального компьютера
    ' с помощью вызова Windows API
    Dim strName As String
    Dim lngChars As Long
    Dim lngRet As Long

    strName = Space(255)
    lngChars = 255

    lngRet = GetComputerName(strName, lngChars - 1)
    If lngRet > 0 Then
        GetThisComputerName = Left(strName, lngChars)
    Else
        GetThisComputerName = "Невозможно получить имя."
    End If
End Function
```

Несмотря на то, что `GetThisComputerName` представляет собой обычную функцию, нужно учесть такие нюансы. Во-первых, нужно быть внимательным при передаче строковых аргументов вызовам API, так как определение строкового значения в API отличается от такового в VBA. Строки API имеют фиксированную длину, в то время как в VBA — переменную. Исходя из этого, в вызовы API передается как сама строка, инициализированная в некоторую длину, так и вспомогательная переменная, сообщающая эту длину. Во-вторых, разбор аргументов в вызовах API производится в обратном порядке, так что вначале будет прочитана длина строки, а затем уже сама строка фиксированной длины.

Совет

Функция VBA `Space()` возвращает строку, состоящую из заданного количества пробелов. Это — идеальный способ инициализации строкового аргумента для вызовов API.

Функциям API требуется в строках дополнительная позиция для хранения маркера конца строки. Именно поэтому при передаче строки длиной в 255 символов мы одновременно сообщаем API, что для работы в ней предоставлено всего 254 символа.

Когда строка возвращается функцией API, можно использовать функцию `Left` для отсеечения символов, находящихся вне заявленной длины строки. Этот фрагмент процедуры обращает внимание еще на один вопрос: большин-

ство вызовов API при успешном завершении возвращают нуль, или ненулевое значение, если возникла какая-либо ошибка.

На рис. 23.1 показан результат вызова этой функции в системе тестирования. Естественно, при запуске на вашем компьютере возвращенное значение будет отличаться.

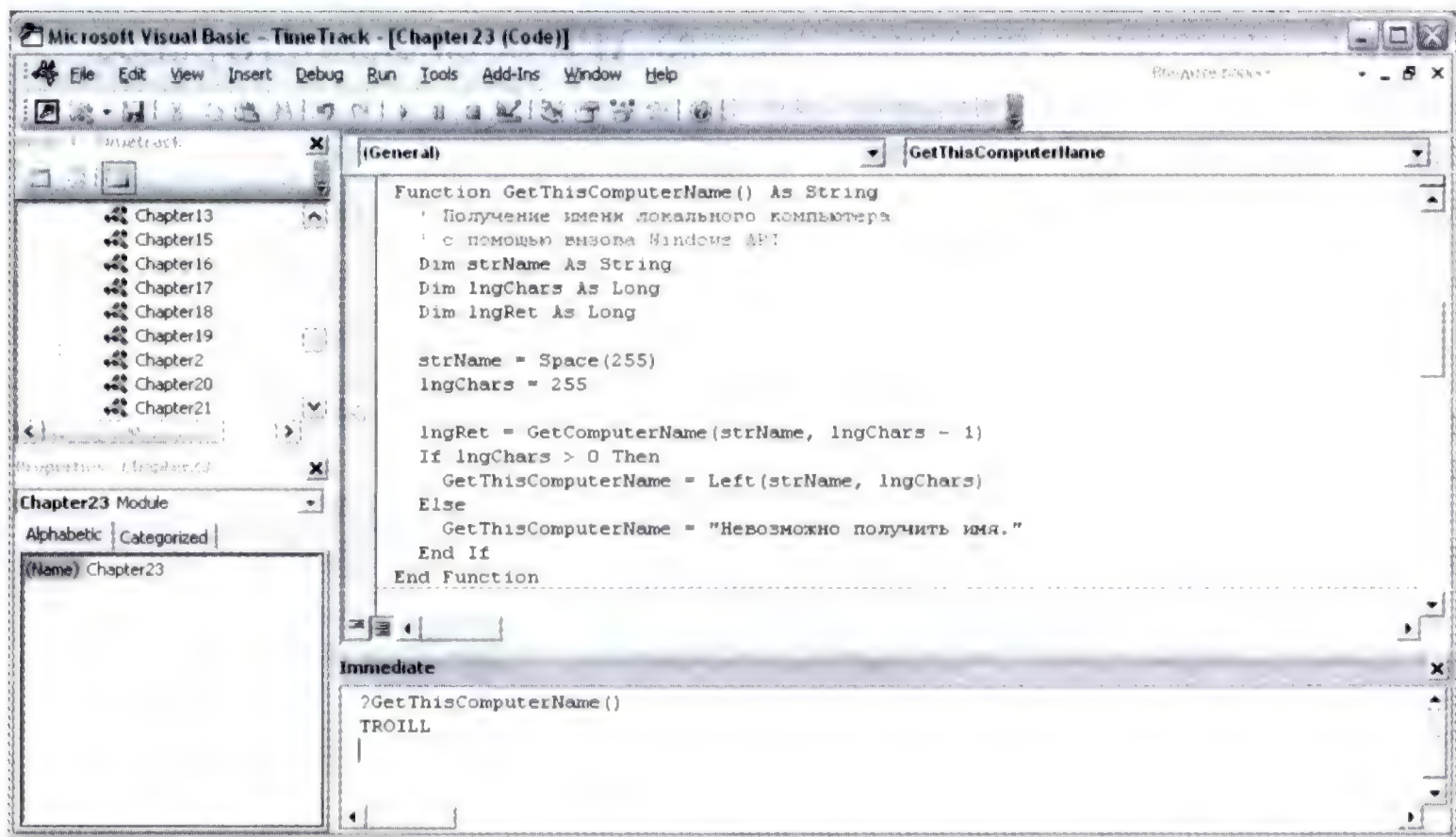


Рис. 23.1. С помощью вызова Windows API получено имя локального компьютера

Вызовы API, которые можно использовать в Access

Интерфейс Windows API содержит тысячи функций и процедур, поэтому в одной главе невозможно представить даже краткий их обзор. Вместо этого мы решим несколько задач, с которыми поможет вам справиться Windows API:

- выясним, запущено ли приложение;
- получим имя текущего пользователя;
- получим имя программы обработки для файла данных.

Как определить, запущено ли приложение

Иногда требуется знать, запущено ли уже некоторое приложение в системе. К примеру, если проводится автоматизация с приложением Excel в качестве сервера автоматизации, перед вызовом процедуры нужно знать, запущена ли в операционной системе хотя бы одна копия этого приложения. Вызов функции API FindWindow вам в этом поможет.

```
Declare Function FindWindow Lib "user32" Alias _
    "FindWindowA" (ByVal lpClassName As String, _
    ByVal lpWindowName As String) As Long

Function IsAppRunning(strClassName As String, _
```



```

strwindowname As String) As Boolean
' Определение, запущено ли приложение,
' по имени класса или заголовку окна
If strClassName = "" Then
    IsAppRunning = (
        FindWindow(vbNullString, strwindowname) <> 0)
Else
    IsAppRunning = (
        FindWindow(strClassName, vbNullString) <> 0)
End If
End Function

```

Функция `FindWindow` извлекает имена классов и заголовки окон и проверяет их на предмет соответствия искомой строке. Если что-то в результате поиска обнаруживается, возвращается порядковая метка этого окна. Порядковые метки — это последовательные номера, которые система Windows присваивает объектам для их отслеживания. Здесь важно отметить, что метка окна никогда не может иметь нулевое значение.

Любое окно на экране идентифицируется операционной системой по его имени класса. Это имя назначается разработчиком программы. К примеру, имя класса программы Excel — `xlMain`, а программы Word — `OpusApp` (*Opus* — это имя пингвина-талисмана изначальной команды разработчиков Word). Если вам не известно имя класса приложения, вы можете выполнить проверку текста заголовка окна. Однако в этом случае вы должны абсолютно точно знать, что отображается на экране. К примеру, при открытии приложения Excel по умолчанию загружается новая рабочая книга, и в этом случае вы можете получить следующие результаты выполнения предложенной процедуры:

```

?IsAppRunning("", "Excel")
False
?IsAppRunning("", "Microsoft Excel - Книга1")
True

```

Функция `FindWindow` ожидает, что если некоторый аргумент не используется, вы будете передавать вместо него пустую строку. Для выполнения этой задачи нулевая строка VBA не подходит, но существует константа `vbNullString`.

Получение имени текущего пользователя

Предположим, что вы создаете программу, отслеживающую, что, где и когда происходит. Важной составляющей этой информации является имя пользователя, проводящего операцию. И здесь снова нам на помощь приходит Windows API.

```

Declare Function GetUserName Lib "advapi32.dll" Alias _
    "GetUserNameA" (ByVal lpBuffer As String, _
    nSize As Long) As Long

Function GetCurrentUserName() As String
' Получение имени текущего пользователя
' с помощью Windows API
Dim strName As String
Dim lngChars As Long

```



```

Dim lngRet As Long

strName = Space(255)
lngChars = 255

lngRet = GetUserName(strName, lngChars - 1)
If lngRet > 0 Then
    GetCurrentUserName = Left(strName, lngChars)
Else
    GetCurrentUserName = "Невозможно получить имя."
End If
End Function

```

Как видите, программа получения имени пользователя практически идентична процедуре извлечения имени компьютера. Единственным отличием является имя вызываемой функции API и название библиотеки, в которой она содержится. Общая схема создания строкового аргумента, передача его длины и проверка возвращаемого значения остались без изменений.

Совет

Вас может удивить, почему так много фактических функций Windows API заканчиваются символом A. Дело в том, что для большинства функций существует одновременно две версии: A и W. Первая из них использует таблицу символов ANSI, а вторая — Unicode. Язык VBA работает только с символами ANSI, поэтому мы используем только одну версию фактических функций.

Получение имени программы обработки для файла данных

Установлено ли в системе приложение Word, или файлы с расширением .doc нужно открывать другой программой, к примеру WordPad? Открывать ли файл .html, который в данное время отображает браузер? Ответ на оба этих вопроса можно найти с помощью вызова функции API FindExecutable. Эта функция сообщает, какое приложение ассоциировано в системе с данным типом файла.

```

Declare Function FindExecutable Lib "shell32.dll" _
    Alias "FindExecutableA" (ByVal lpFile As String, _
    ByVal lpDirectory As String, ByVal lpResult As String) _
    As Long

Function GetMatchingApp(strDataFile As String, _
    strDir As String) As String
    ' Поиск программы открытия
    ' заданного типа файла

    Dim strApp As String
    Dim lngRet As Long

    strApp = Space(260)
    lngRet = FindExecutable(strDataFile, strDir, strApp)

```



```

If lngRet > 32 Then
    GetMatchingApp = strApp
Else
    GetMatchingApp = "Подходящее приложение не найдено."
End If
End Function

```

Этой функции нужно передать имя файла данных (который обязательно должен существовать) и имя рабочего каталога. Результатом функции будет имя и путь к приложению, которое обычно запускается, когда вы в Проводнике дважды щелкаете на имени файла данных, например:

```

?GetMatchingApp("c:\Clients.txt", "c:\temp")
C:\WINDOWS\system32\notepad.exe

```

Обратим внимание на магическое число 260, соответствующее длине строки, которая должна содержать возвращаемое значение. Это — максимальная длина пути вместе с именем файла, допустимая в Windows. Как видите, функция FindExecutable нарушает одно из общих правил: при ошибке она возвращает не нуль, а число, меньшее 32.

Когда следует использовать Windows API

Самый сложный вопрос при работе с Windows API — не *как* его использовать, а *когда*. Справиться с этой задачей вам помогут следующие рекомендации.

- Используйте Windows API только тогда, когда без него обойтись невозможно.
- Используйте Windows API для манипуляций приложениями, отличными от Access.
- Используйте Windows API для извлечения информации из операционной системы Windows.
- Используйте Windows API для взаимодействия с объектами, недоступными в VBA (например, с сообщениями локальной сети).

Если вы серьезно заинтересуетесь расширенными средствами программирования на языке VBA, вам помогут специализированные книги. Хорошей отправной точкой может послужить книга Дэна Эпплмана (Dan Appleman) *Visual Basic Programmer's Guide to the Win32API*, выпущенная издательством Sams в 1999 году.



Приложение

Обзор языка SQL

Введение в SQL

VBA является не единственным языком, который “понимает” программа Access. Язык структурированных запросов *SQL* (Structured Query Language) — “родной” язык реляционных баз данных, поэтому программа Access не может его не понимать. Более конкретно, приложение Access “разговаривает” на диалекте этого языка, известном как *Jet SQL* (иногда его еще называют *Access SQL*, хотя такой термин ошибочен по той причине, что SQL служит для общения не с программой Access, а с механизмом *Jet*). В приложении будет изложена общая информация о языке *Jet SQL*, а также описан синтаксис его инструкций. Это краткое приложение можно рассматривать как введение в основные инструкции; к нему нельзя относиться как к полному справочнику по инструкциям и правилам языка SQL.

Язык SQL является стандартом индустрии управления реляционными базами данных. На рынке представлена масса реляционных баз данных, и каждая из них имеет свою версию языка SQL. Как уже говорилось, в базах данных Access используется *Jet SQL*. Изначально язык SQL (произносится как “эс-кю-эль”) носил название

В ЭТОМ ПРИЛОЖЕНИИ

Введение в SQL.....	395
Структура и синтаксис языка SQL	396
Извлечение данных с помощью инструкции SELECT.....	398
Изменение данных с помощью инструкции UPDATE.....	403
Удаление записей с помощью инструкции DELETE	404
Добавление записей с помощью инструкции INSERT INTO.....	405
Создание таблиц с помощью инструкции SELECT INTO.....	406
Создание перекрестного запроса с помощью инструкции TRANSFORM.....	406

SEQL (произносится как “сиквел”) (Structured English Query Language), однако вследствие юридического конфликта по поводу этого имени он был переименован.

До сих пор еще можно услышать термин “сиквел”: как известно, привычки живучи. И это еще раз доказывает, что данный язык пережил поколения программистов. Старое название — всего лишь дело привычки, а не какой-либо акт протеста.

Совет

Многие стандарты, в том числе и стандарты SQL, адаптируются, координируются и управляются Американским национальным институтом стандартизации (ANSI). Более подробную информацию о стандартах SQL можно найти на узле www.ansi.org.

Предупреждение

Несмотря на существование стандарта SQL, ни один продукт не придерживается его в полном объеме. В этом приложении мы обсудим только версию Jet SQL; приведенные здесь примеры выражений могут не работать в других базах данных.

Язык JetSQL используют для прямого взаимодействия с данными средствами механизма Jet. Все его инструкции можно формально разделить на две части:

- **Язык управления данными (DML).** Этот язык используется для извлечения, изменения и удаления существующих данных, а также для добавления новых данных.
- **Язык определения данных (DDL).** Этот язык используется для управления объектами в базе данных, в приложении мы не будем его рассматривать.

Предупреждение

На протяжении всего приложения будут использованы примеры выражений над данными и таблицами базы данных примеров TimeTrack. Эти примеры носят исключительно демонстративный характер. Не запускайте их на выполнение в базе данных примеров, так как это может навредить работоспособности примеров, приведенных в основных главах книги. Если вы решили изучить работу этих выражений, создайте копию базы данных TimeTrack и экспериментируйте с этой копией.

Структура и синтаксис языка SQL

Язык SQL легко изучить и использовать, если знать правильный синтаксис передачи данных. Большей своей частью язык SQL состоит из *ключевых слов* — отдельных слов, имеющих predetermined значение. Большинство ключевых слов могут содержать аргументы подобно функциям и процедурам. Ключевое слово с аргументом называют *предложением*.

Комбинация предложений, образующая запрос механизма базы данных, называется *выражением* или *инструкцией*. Выражение может состоять всего из одного предложения, однако обычно их несколько.

Основная общая форма выражения имеет следующую конструкцию:

действие список_полей FROM источник_данных

где *действие* — одно из множества ключевых слов, определяющих цель выражения, *список_полей* — множество полей, участвующих в действии, а *источник_данных* — таблица, в которой эти данные хранятся.

Объем записей можно ограничить, если ввести в эту форму предложение WHERE, в результате чего выражение принимает следующий вид:

действие список_полей FROM источник_данных WHERE условие

В табл. 1 перечислены наиболее часто используемые ключевые слова действий языка SQL, а табл. 2 содержит некоторые ключевые слова, выступающие в роли *предикатов* и дополняющие выражение.

Таблица 1. Ключевые слова действий языка SQL

Ключевое слово	Назначение	Синтаксис
SELECT	Извлечение данных	SELECT <i>список_полей</i> FROM <i>источник_данных</i>
UPDATE SET	Изменение данных	UPDATE <i>источник_данных</i> SET <i>столбец</i> = <i>выражение</i>
DELETE	Удаление записей	DELETE FROM <i>источник_данных</i>
INSERT INTO	Вставка (или добавление) записей в существующую таблицу	INSERT INTO <i>целевая_таблица</i> SELECT <i>источник_данных</i>
SELECT INTO	Копирование данных и структуры существующей таблицы в новую	SELECT <i>список_полей</i> INTO <i>новая_таблица</i> FROM <i>источник_данных</i>

Таблица 2. Ключевые слова и предикаты дополнительной информации

Ключевое слово	Назначение	Синтаксис
ALL	Предикат, обозначающий действие над всеми записями таблицы или возвращение всех записей	SELECT ALL FROM <i>источник_данных</i>
DISTINCT	Предикат, возвращающий только уникальные значения заданных полей	SELECT DISTINCT <i>список_полей</i> FROM <i>источник_данных</i>

Окончание табл. 2

Ключевое слово	Назначение	Синтаксис
FROM	Идентифицирует источник данных	FROM <i>источник_данных</i>
TOP	Ограничивает количество возвращаемых записей или записей, над которыми выполняется действие	TOP <i>количество</i>
WHERE	Условие, ограничивающее отбор записей	WHERE <i>условное_выражение</i>
GROUP BY	Группирует записи по сходным значениям	GROUP BY <i>список_полей</i>
ORDER BY	Упорядочивает записи по полю или полям	ORDER BY <i>список_полей</i>
HAVING	Применяет критерий к итоговому столбцу	HAVING <i>условие</i>

Извлечение данных с помощью инструкции SELECT

В этой книге часто использовалась инструкция SELECT для извлечения данных с целью просмотра или дальнейшего использования. Инструкция SELECT не выполняет никаких действий над самим источником данных.

В то же время инструкция SELECT достаточно гибка, а ее структура может быть довольно сложной:

```
SELECT ALL|DISTINCT|TOP список_полей | *  
FROM источник_данных  
[WHERE условие]  
[GROUP BY столбец1 [, столбец2 ...]]  
[HAVING условие]  
[ORDER BY столбец1 [ASC|DESC] [, столбец2 ... [ASC|DESC] ]
```

Если опустить компонент ALL|DISTINCT, SQL предполагает наличие первого из них. Аналогично, в предложении ORDER BY по умолчанию предполагается наличие ключевого слова ASC. Символ звездочки предполагает отбор всех полей в источнике данных, однако этого лучше избегать, поскольку не все поля несут нужную смысловую нагрузку. К тому же, чем больший объем данных вы извлекаете, тем медленнее будет выполняться запрос. Предложение FROM — единственное, являющееся обязательным.

Простейшая инструкция SELECT не использует необязательных предложений:
SELECT * FROM *источник_данных*

Аргумент *список_полей* представляет собой, как и следует из названия, список имен полей источника данных, разделенных запятыми:

```
SELECT поле1 [AS псевдоним] [, поле2 [AS псевдоним] ...]
```

где *поле1*, *поле2* и т.д. представляют имена полей. В этом списке можно перечислить столько полей, сколько требуется иметь в результатах запроса. Предложение AS служит модификатором имен полей. Полю, которое в источнике данных имело некоторое имя, можно в результатах запроса присвоить *псевдоним*. К примеру, в следующей инструкции имена Lname и Fname преобразуются в LastName и FirstName соответственно:

```
SELECT Lname AS LastName, Fname AS FirstName FROM Employees
```

Предикаты SQL

Предикаты ALL, DISTINCT и TOP определяют состав извлекаемых записей. Предикат ALL используется по умолчанию и указывает на извлечение всех записей источника данных. Предикат DISTINCT указывает на отбор тех записей, в которых перечисленные в списке поля имеют уникальные значения. К примеру, в следующей инструкции будут отображены записи с уникальными именами и фамилиями:

```
SELECT DISTINCT Lname AS LastName, Fname AS FirstName FROM Employees
```

Другими словами, если бы в источнике данных содержалось две записи с именем Иван Демидов, в результирующий набор записей попала бы только одна (как правило, первая из них).

Предупреждение

Предикат DISTINCT возвращает набор данных, который не может быть обновлен, поэтому не используйте его при изменении данных.

Различие между предикатами DISTINCT и DISTINCTROW состоит в следующем: первый следит за уникальностью множества перечисленных в запросе полей, а второй — за уникальностью значений всех полей источника данных.

Предикат TOP позволяет ограничить число записей результирующего набора данных. Он имеет следующий синтаксис:

```
SELECT TOP n [PERCENT] столбец1[, столбец2...]
```

где *n* — количество записей, которые мы хотим получить. Для примера, следующая инструкция извлекает первые 10 записей таблицы Employees:

```
SELECT TOP 10 Lname, Fname FROM Employees
```

Ключевое слово PERCENT интерпретирует значение *n* как процент от общего числа записей, а не их абсолютное количество. Следующая инструкция отбирает 10% всех записей таблицы Employees:

```
SELECT TOP 10 PERCENT Lname, Fname FROM Employees
```


Первый пример всегда будет возвращать 10 записей, в то время как количество записей, возвращаемых вторым, будет варьироваться в зависимости от объема источника данных. При наличии в нем 50 записей второе выражение вернет первые пять; если же их тысяча — то первые 100.

Предложение FROM

Предложение FROM является обязательным, и его полная форма имеет следующий вид:

```
FROM источник_данных | таблица_один СВЯЗЬ таблица_много ON _
таблица_один.первичный_ключ = таблица_много.вторичный_ключ
```

где *таблица_один* и *таблица_много* указывают на две таблицы, участвующие в отношении “один ко многим”. Более сложная форма позволяет извлекать данные из связанных таблиц. Аргумент *СВЯЗЬ* идентифицирует тип связи между таблицами: INNER JOIN, LEFT или RIGHT. Существуют и другие типы отношений, однако только эти три поддерживаются языком Jet SQL. Выражение в следующем примере извлекает все записи о клиентах, а также все проекты каждого клиента (это обеспечивается условием `Clients.ClientID = Clients.ClientIDFK`):

```
FROM Clients INNER JOIN Projects ON Clients.ClientID = _
Clients.ClientIDFK
```

Если некоторый клиент не имеет проектов, его запись не будет извлечена ввиду применения внутреннего отношения INNER JOIN. Аналогично, если проект не относится ни к какому клиенту (что на самом деле невозможно), этот проект не появится в результирующем наборе данных.

Для того чтобы вернуть всех клиентов, даже тех, у которых нет проектов, можно использовать отношение LEFT. Аналогично, чтобы вернуть все проекты, даже те, которым не назначен клиент, можно использовать отношение RIGHT. Отношение LEFT возвращает все записи из левой таблицы отношения (*таблица_один*) и только соответствующие им записи из правой (*таблица_много*); отношение RIGHT возвращает все записи из правой таблицы и только соответствующие им из левой.

Предложение WHERE

Часто состав извлекаемых данных (или тех, над которыми выполняются операции) зависит от некоторого заданного критерия. Для этого и предназначено предложение WHERE языка SQL — ограничивать объем извлекаемых, обновляемых или удаляемых данных.

Это предложение имеет следующий синтаксис:

```
WHERE условное_выражение
```

где *условное_выражение* может принимать множество форм, однако суть состоит в том, что в результат запроса включаются только те записи, которые

удовлетворяют этому условию. В простейшем случае условным выражением может быть операция сравнения значения некоторого поля с константой:

```
SELECT *
FROM Clients
WHERE ClientID = 1
```

Эта инструкция вернет только те записи из таблицы `Clients`, в которых поле `ClientID` содержит единицу. В то же время, этот аргумент может быть и чрезвычайно сложной комбинацией условий, объединенных операторами `AND` и `OR`. Для примера предположим, что нам нужны все проекты конкретного клиента, начавшиеся в январе 2004 года. В данном случае мы можем использовать следующую инструкцию:

```
SELECT *
FROM Projects
WHERE ClientID = 1 And StartDate Between #1/1/2004# And #31/1/2004#
```

Обратите внимание, что константы дат должны обрамлять знаки диеза (`#`), а константы строк — кавычки. Предложение `WHERE` является достаточно гибким, чтобы обрабатывать критерии, содержащие столбцы, не включенные в список полей выражения (главное, чтобы они принадлежали указанному в выражении источнику данных). К примеру, состав извлекаемых полей в предыдущем выражении можно ограничить только информацией о клиенте, но при этом объем извлекаемых записей не изменится:

```
SELECT ClientID, ProjectName
FROM Projects
WHERE ClientID = 1 And StartDate Between #1/1/2004# And #31/1/2004#
```

Чтобы увидеть записи, содержащие значение `NULL`, это нужно явно указать в выражении. К примеру, следующее выражение:

```
SELECT ClientID, ProjectName
FROM Projects
WHERE ClientID = 1 And StartDate Between #1/1/2004# And #31/1/2004#
Or StartDate Is Null
```

вернет даже те записи первого клиента, дата начала проекта в которых не определена (равна `NULL`).

Предложение ORDER BY

Одного извлечения записи не всегда достаточно — иногда важную роль играет порядок записей. Предложение `ORDER BY` может осуществлять сортировку по текстовым и числовым данным, а также по датам. Не пытайтесь применять для упорядочивания другие типы данных, так как эта попытка безоговорочно приведет к ошибке. Для задания порядка сортировки используйте предложение `ORDER BY` в следующей форме:

```
ORDER BY столбец1 [ASC | DESC] [, столбец2 [ASC | DESC] ...]
```

Компонент `ASC` | `DESC` определяет вид сортировки: по возрастанию или по убыванию соответственно. Если этот компонент опущен, по умолчанию принимается сортировка по возрастанию. К примеру, следующая инструкция

возвращает проекты первого клиента, отсортированные по дате начала проекта, в порядке возрастания:

```
SELECT ClientID, ProjectName
FROM Projects
WHERE ClientID = 1 And StartDate Between #1/1/2004# And #31/1/2004#
ORDER BY StartDate
```

Подобно предложению WHERE, сортировку можно проводить по столбцу, явно не перечисленному в списке полей инструкции SELECT, но имеющемуся в источнике данных. Учтите, что порядок полей сортировки установлен слева направо.

Предложение GROUP BY

Подобно функции агрегирования, предложение GROUP BY определяет группы и иногда выполняет вычисление итогов по группе. К примеру, группу можно определить для возвращения итоговых сумм по каждому из клиентов, не возвращая строк отдельных задач.

Синтаксис предложения GROUP BY следующий:

```
GROUP BY столбец1 [, столбец2...]
```

где *столбец1*, *столбец2* и т.д. являются ссылками на фактические столбцы или вычисляемые поля. Единственное условие участия вычисляемых полей в этом предложении — отсутствие в их выражениях функций и констант агрегирования. Еще одним условием является то, что все поля в инструкции SELECT должны присутствовать в следующем виде:

- как аргументы функций агрегирования;
- в предложении GROUP BY — как ссылки на поля.

На первый взгляд совершенно неочевидно, что каждый столбец в результирующем наборе записей либо определяет группу, либо участвует в вычислениях по ней. Здесь важно заметить, что столбец не обязан быть частью списка полей, чтобы быть включенным в предложение GROUP BY — ему достаточно просто находиться в источнике данных инструкции. Подобно предложению ORDER BY, здесь группы формируются слева направо. Для примера предположим, что требуется сгруппировать записи о проектах клиентов, чтобы впоследствии их подсчитать. В этом случае можно использовать следующую инструкцию:

```
SELECT Projects.ClientID, Projects.ProjectName, _
    Count(Projects.StartDate) AS CountOfStartDate
FROM Projects

WHERE Projects.ClientID = 1 AND (Projects.StartDate _
    Between #1/1/2004# And #31/01/2004#)
GROUP BY Projects.ClientID
```

Обратите внимание, что каждый столбец в инструкции SELECT является либо частью функции агрегирования (в данном случае функции Count) или перечислен в предложении GROUP BY.

Предложение HAVING

Предложение HAVING ограничивает результаты группировки, произведенной предложением GROUP BY. В то же время при наличии в инструкции группировки совсем не обязательно использование предложения HAVING. Здесь самое важное — запомнить, что механизм Jet применяет предложение HAVING уже после группировки данных. Это значит, что вы можете его использовать для отсекаания определенных записей уже после группировки, что может оказаться полезным при подведении итогов по группам и последующем применении к результатам некоторого условия.

Предложение HAVING имеет предельно простой синтаксис:

HAVING *условие*

Аргумент *условие* может представлять собой одно или несколько условий, скомбинированных операторами OR и AND в более сложную форму:

HAVING *выражение1* AND | OR *выражение2*

Следующая инструкция — хороший пример использования предложения HAVING для ограничения групп:

```
SELECT EmployeeID, DateWorked, Sum(Hours) AS SumOfHours
FROM Timeslips
GROUP BY EmployeeID, DateWorked
HAVING Sum(Hours) > 8
```

В этой инструкции группировка основана на полях EmployeeID и DateWorked. В то же время предложение HAVING будет возвращать только те записи, в которых сумма значений поля Hours по некоторой дате (внутренняя часть группировки) больше 8 часов. Обратите внимание на то, что в предложении HAVING оценивается результат функции агрегирования Sum(Hours), участвующей в списке полей инструкции SELECT. Такая конструкция необязательна, однако чаще всего используют именно ее.

Совет

Предложения WHERE и HAVING используются для ограничения результатов сгруппированного набора записей, однако работают они по-разному. Первый исключает записи до группировки, а второй — после. Часто результатом применения обоих предложений становится один и тот же результат, однако это не должно вас вводить в заблуждение, что эти конструкции взаимозаменяемы.

Изменение данных с помощью инструкции UPDATE

Инструкция UPDATE языка SQL аналогична запросу обновления, выполняемому в интерфейсе пользователя Access для изменения существующих данных. Эта инструкция использует следующий синтаксис:


```
UPDATE источник_данных  
SET столбец1 = выражение1[, столбец2 = выражение2...]  
[WHERE условие]
```

Принимая во внимание назначение этой инструкции, в ней нельзя использовать предложения группировки и сортировки (GROUP BY, HAVING и ORDER BY). К тому же инструкция UPDATE не возвращает результирующий набор данных, однако Access отображает сообщение о количестве записей, подлежащих обновлению, и запрашивает у пользователя подтверждения операции. К примеру, следующая простая инструкция UPDATE изменяет регионы всех записей, содержащих значение Москва в поле City, на Центральный:

```
UPDATE Clients SET State = "Центральный"  
WHERE City = "Москва"
```

Совет

Перед тем как изменять данные такими запросами, как UPDATE, рекомендуется создавать копию источника данных. При этом если что-то пойдет не так, у вас под рукой всегда будет копия неизмененных данных. После проверки корректности проведенных изменений эту временную копию можно удалить.

Изменение данных обычно не сказывается на их целостности, если, конечно, вы не изменяете значения первичного и вторичного ключей (чего делать вообще не рекомендуется). В то же время изменения должны отвечать свойствам и правилам корректности, установленным для полей и таблиц. При запуске операции обновления данных из программы обязательно включайте обработку этих потенциальных ошибок.

Удаление записей с помощью инструкции DELETE

Инструкция DELETE удаляет записи в источнике данных точно так же, как запрос на удаление, создаваемый в интерфейсе пользователя Access. Самой важной особенностью этой инструкции является то, что запись удаляется целиком. По этой причине в синтаксисе инструкции DELETE отсутствует список полей. Ее синтаксис следующий:

```
DELETE  
FROM источник_данных  
[WHERE условие]
```

Естественно, вы можете указать поля, и это не приведет к ошибке, но в то же время будет абсолютно бессмысленным. Инструкция DELETE поймет символ подстановки *, однако он тоже не имеет смысла, так как в любом случае запись удаляется целиком. Следующая инструкция удалит запись сотрудника Руслан Пилипейко из таблицы сотрудников Employees. Как уже говорилось, не рекомендуется выполнять эту инструкцию на базе данных примеров, так как это может повлиять на результаты примеров в основных главах книги.


```
DELETE
FROM Employees
WHERE EmployeeID = 3
```

Добавление записей с помощью инструкции INSERT INTO

Задача добавления записей встает довольно часто. Для ее решения можно использовать инструкцию `INSERT INTO` языка SQL. При этом данные из одной таблицы копируются в другую, или в некоторой таблице создается новая запись. При копировании записей из одной таблицы в другую используется следующий синтаксис:

```
INSERT INTO таблица_приемник
SELECT таблица_источник
```

где *таблица_приемник* — это таблица, в которую копируются данные, а *таблица_источник* — та, из которой эти записи берутся. Эта инструкция поддерживает предложения `WHERE` и `ORDER BY`.

Инструкция `INSERT INTO` не создает новую таблицу, поэтому таблица-приемник должна существовать на момент выполнения этой инструкции. К тому же в таблице-приемнике должны существовать вставляемые столбцы — SQL не создает новых столбцов.

Предупреждение

Применение в инструкции `INSERT INTO` символа звездочки может привести к ошибке, если обе таблицы используют поля автонумерации `AutoNumber` с целью создания первичного ключа. Причиной этой ошибки будет то, что операция вставки будет пытаться дублировать значения первичного ключа в таблице-приемнике, что недопустимо.

Для добавления записей (по одной) в таблицу используют следующий синтаксис:

```
INSERT INTO таблица [столбец1[, столбец2...]]
VALUES [значение1[, значение2...]]
```

Для успешного использования этой формы инструкции следует учесть следующие правила:

- Ссылки на столбцы не обязательны; когда они опущены, значения должны передаваться для всех полей таблицы по порядку.
- Значения в предложении `VALUES` должны иметь тот же порядок, что и в списке полей таблицы инструкции `INSERT INTO`. Этот порядок может отличаться от заложенного в исходной конструкции таблицы.

Следующая простая инструкция

```
INSERT INTO Employees (LastName, FirstName)
VALUES ('Демидов', 'Иван')
```


добавит новую запись для сотрудника Иван Демидов в таблицу Employees. Хотелось бы особо отметить, что поле автонумерации EmployeeID в инструкции отсутствует, поскольку это поле генерируется механизмом Jet автоматически. Если вы попытаетесь задать значение для поля автонумерации в инструкции вручную, механизм Jet вернет сообщение об ошибке.

Создание таблиц с помощью инструкции SELECT INTO

Язык SQL позволяет манипулировать не только существующими данными. Вы можете создавать новые таблицы, и для этого использовать инструкцию SELECT INTO. Вместо копирования данных из одной таблицы в другую, как в случае инструкции INSERT INTO, инструкция SELECT INTO копирует данные из таблицы-источника в новую, которую сама же и создает.

Для создания новой таблицы на основе существующих данных используйте следующий синтаксис инструкции SELECT INTO:

```
SELECT * | столбец1[, столбец2...] INTO новая_таблица
FROM источник_данных
```

Инструкция SELECT INTO поддерживает предложения WHERE, GROUP BY, HAVING и ORDER BY для ограничения и организации данных в новой таблице.

Для копирования в новую таблицу только структуры старой (без данных) используют следующий синтаксис:

```
SELECT * INTO новая_таблица
FROM источник_данных
WHERE False
```

Так как условие WHERE представлено константой False, оно никогда не выполняется, вследствие чего ни одна запись не будет скопирована в новую таблицу. Эта инструкция копирует только структуру полей источника данных, в новой таблице она не формирует первичных ключей и не создает индексов.

Предупреждение

Инструкция SELECT INTO заменяет таблицу с таким же именем, если таковая существует. Обязательно проверяйте существование одноименных таблиц *перед* запуском на выполнение инструкции SELECT INTO и поступайте соответственно результатам. Обычной реакции на сообщение о том, что некоторая таблица будет удалена, недостаточно, так как даже при отмене операции Access ее все равно удалит.

Создание перекрестного запроса с помощью инструкции TRANSFORM

Перекрестные запросы являются отличным методом представления больших объемов данных в осмысленном и удобном для восприятия формате. Эти запросы подытоживают данные по категориям, как правило, используя для

этого функции агрегирования. Общим правилом является следующее: этот запрос должен иметь, по крайней мере, один заголовок столбца и один итоговый столбец.

Перекрестные запросы выполняет инструкция TRANSFORM языка SQL, имеющая следующий синтаксис:

```
TRANSFORM агрегация
SELECT инструкция
PIVOT столбец
```

где *агрегация* — одна или несколько функций агрегирования, используемые для вычисления столбца, *инструкция* — это любая допустимая инструкция SQL, а *столбец* идентифицирует заголовок столбца (или заголовки столбцов). Инструкция TRANSFORM не поддерживает предложение HAVING.

Совет

Вначале перекрестные таблицы немного пугают, однако существует простой способ получить желаемый результат:

1. Начните с инструкции SELECT, группирующей столбцы с помощью функции агрегирования.
2. Вставьте ключевое слово TRANSFORM перед инструкцией SELECT и переместите функцию агрегирования из конструкции SELECT в конструкцию TRANSFORM.
3. Вставьте после конструкции SELECT ключевое слово PIVOT и переместите столбцы из предложения GROUP BY в новое предложение PIVOT.

Невозможно гарантировать, что результаты будут в точности теми, каких вы ожидаете, но, по крайней мере, они будут очень близки. После того как у вас будет готова базовая структура, вам будет легче добавлять заголовки столбцов и соответствующие функции агрегирования.

В следующей инструкции заложена масса информации:

```
TRANSFORM Sum([HourlyRate]*[Hours]) AS Total
SELECT Clients.Client, Projects.ProjectName
FROM Clients INNER JOIN Projects ON Clients.ClientID =
Projects.ClientID
INNER JOIN Tasks ON Projects.ProjectID = Tasks.ProjectID
INNER JOIN Timeslips ON Tasks.TaskID = Timeslips.TaskID
GROUP BY Clients.Client, Projects.ProjectName
PIVOT Projects.StartDate
```

В результате выполнения этой инструкции будет получен список клиентов и проектов с количеством часов работы в столбцах дат начала. Другими словами, вы сможете узнать, какая сумма была начислена клиенту по каждому из его проектов, и быстро отыскать дату его начала. Так как этот запрос извлекает данные из более чем одной таблицы, в ссылки на поля во избежание путаницы включены и имена таблиц. В частности, это применялось в отношении поля ClientID таблиц Clients и Projects.

Предметный указатель

A

ADO, 277
 курсоры, 285
 динамический, 285
 ключевой, 285
 последовательного доступа, 285
 статический, 285
 объект Command, 283
 выполнение команд, 284
 создание объекта, 283
 объект Connection
 метод Close, 282
 метод Open, 281
 метод OpenSchema, 338
 методы обработки транзакций, 310
 свойства, 279
 объект Recordset, 222; 284
 коллекция Fields, 298
 метод AddNew, 303
 метод CancelUpdate, 309
 метод Delete, 305
 метод Find, 299
 метод Seek, 301
 метод Update, 303; 306
 методы перемещения указателя, 295
 свойства EOF и BOF, 296
 фильтрация записей, 287
ADOX, 315
 объект Catalog, 317
 коллекция Groups, 322
 коллекция Users, 322
 метод GetObjectOwner, 324
 метод SetObjectOwner, 324
 объект Column, 317
 объект Key, 321
 объект Table, 317
 средства безопасности, 322
ANSI, 101
ASCII, 101

O

OneNote 2003, 27

S

SQL, 29; 395
 DDL, 396
 DML, 396
 Jet SQL, 395
 инструкции, 397
 DELETE, 404
 INSERT INTO, 405
 SELECT, 398
 SELECT INTO, 406
 TRANSFORM, 407
 UPDATE, 403
 ключевые слова, 397
 предикаты, 399
 предложения, 396
 FROM, 400
 GROUP BY, 402
 HAVING, 403
 ORDER BY, 401
 WHERE, 400

V

VBA, 25; 30
VBE, 35
 Обозреватель объектов, 154
 панель инструментов, 36
 панель справки, 43
 режим прерываний, 76
 установка контрольных точек, 77
 функция IntelliSense, 43; 79

W

Windows API, 387
 функция FindExecutable, 392
 функция FindWindow, 390

Х

XML, 373

файлы XML, 373

файлы XSD, 374

файлы XSL, 374

А

Автоматизация приложений, 359

Б

Библиотека DAO, 225

Библиотека типов, 360

Блокировка записей

оптимистическая, 331; 333

пессимистическая, 331; 336

Блокировка страниц, 332

Буфер данных, 204

В

Время жизни

объекта, 169

переменных и констант, 163; 168

приложения, 169

уровень модуля, 170

уровень процедуры, 169

Вызов API, 388

ГГлобальный идентификатор GUID,
338**Д**

Дескриптор, 374

И

Идентификатор, 62

Имя переменной, 55

Инструкции VBA, 65

Else, 121

ElseIf, 122

Input #, 351

Line Input #, 351

Option Base, 137

Print #, 354

Randomize, 95

ReDim, 141

Set, , 145

Write #, 353

К

Клиентское приложение, 359

Ключевые слова

And, 121

ByVal, 72

End Function, 67

End Sub, 66

Function, 67

Me, 185

New, 52; 146

Not, 121

Nothing, 173

Or, 121

Preserve, 141

Sub, 66

Коллекции, 263

Комментарии, 48

Конкуренция, 331

Константы, 51

встроенные, 56

константа даты, 91

объявление, 55

М

Макрос, 28

Массив, 135

верхняя граница, 136

динамический, 141

значение элемента, 138

индекс, 136

многомерный, 140

нижняя граница, 136

фиксированной длины, 141

Метка, 74; 131

файла, 345

Методы

ExportXML, 374

ImportXML, 379
табличного сканирования, 302
Механизм Jet, 188
Модификаторы, 69
Private, 69
Public, 69
Модуль
класса, 39
объекта, 38
объектный, 247
стандартный, 38; 247

О

Область видимости, 163
глобальный уровень, 164; 166
локальный уровень, 164
уровень модуля, 164; 165
Обработка ошибок, 73
Объектная модель, 152
Access, 263
ADO, 278
ADOX, 316
ссылка на объектную модель, 153
Объекты
AccessObject, 266
Application, 363
CurrentProject, 180
DoCmd, 159; 177
метод ApplyFilter, 187
метод Close, 178; 250
метод MoveSize, 182
метод OpenForm, 177; 184
класс объектов, 144
коллекция, 150
AllForms, 180
метод SetFocus, 185
метод класса, 144; 148
процедура свойства, 157
свойства объектов, 146
свойство Recordset, 289
свойство по умолчанию, 151
создание собственных объектов,
155
член класса, 154
экземпляр класса, 144
Ограничители, 101
Операторы VBA, 62

Case Else, 186
Const, 55
Dim, 52
Exit Do, 130
Exit For, 128
GoTo, 130
If, 120
If...Then...Else, 120
On Error, 73
Option Explicit, 52
Select Case, 122
слияния, 70

Отчет

группировка данных, 257
метод OpenReport, 248
обработка ошибок, 254
объект GroupLevel, 258
открытие отчета, 248
свойство OpenArgs, 250
сортировка, 253
фильтрация, 253

П

Переменная, 51
имя, 55
инициализация, 85
объекта, 143
объявление, 52
простая, 143
скалярная, 143
статическая, 172
Позднее связывание, 364
Поле, 63
Процедуры, 40; 65
аргументы, 69
передача по значению, 72
передача по ссылке, 71
подпрограмма, 66
функция, 67

Р

Разветвление, 119
Раннее связывание, 364

С

Серверное приложение, 359

Синтаксис ссылок, 62

Синтаксис языка, 51

Словарь данных, 326

События, 158

 Activate, 203; 205

 AfterDelConfirm, 204

 AfterInsert, 204

 AfterUpdate, 200

 BeforeDelConfirm, 204

 BeforeInsert, 204

 BeforeUpdate, 200

 Change, 200

 Close, 203; 205

 Current, 203

 Deactivate, 203; 205

 Delete, 204

 Dirty, 200

 Enter, 197

 Error, 254

 Exit, 197

 Format, 205

 GotFocus, 197

 KeyDown, 200

 KeyPress, 200

 KeyUp, 200

 Load, 203

 LostFocus, 197

 NoData, 205

 NotInList, 202

 Open, 203; 205

 Page, 206

 Print, 205

 Resize, 203

 Retreat, 205

 Unload, 203

 процедура обработки, 158

Соглашение об именах, 46

 венгерское соглашение, 47

 естественная система имен, 47

 префиксы объектов, 47

 префиксы переменных, 48

Создание

 индексов, 319

 отношений, 320

 таблицы, 317

Спецификатор, 62

Схематический набор данных, 291; 339

Счетчик цикла, 125

Т

Текстовое поле

 свободное, 235

Типы данных

 Boolean, 59

 Byte, 60

 Currency, 60

 Date, 60

 Decimal, 60

 Double, 60

 Integer, 60

 Long, 60

 Object, 61

 Single, 61

 String, 61

 Variant, 52; 61

Транзакция, 309

У

Указатель текущей записи, 295

Ф

Фокус, 197

Форма

 закрытие, 178

 модуль, 179

 открытие, 177

 отслеживание фокуса, 233

 подчиненная, 238

 проверка загрузки, 182

 проверка существования, 180

 свойство OpenArgs, 184

 события

 Error, 188

 Load, 183

 Open, 186

 элементы управления

 группа переключателей, 236

 свойство Дополнительные

 сведения, 239

 текстовое поле, 231

Функции

- FindWindow, 390; 392
- OpenForm, 44

Функции VBA

- Abs, 94
- Asc, 100
- CBool, 81
- CByte, 83
- CDate, 83
- Chr, 101
- CInt, 84
- CreateObject, 362
- CStr, 85
- CVar, 85
- Date, 88
- Date\$, 88
- DateAdd, 89
- DateDiff, 89
- DatePart, 91
- DateSerial, 91
- DateValue, 92
- Day, 93
- Ddb, 96
- EOF, 350
- Format, 105
- FreeFile, 346
- Fv, 96
- GetObject, 363
- Hour, 93
- Input, 349
- InputBox, 87; 112
- Int, 94
- IPmt, 97
- IsArray, 110
- IsDate, 110
- IsEmpty, 110
- IsError, 111
- IsMissing, 111
- IsNull, 111
- IsNumeric, 111
- IsObject, 111
- Lcase, 102
- Left, 102
- Len, 102
- LOF, 350
- LTrim, 104

- Mid, 102
- Minute, 93
- Month, 93
- MsgBox, 87; 112
- Now, 87
- nPer, 97
- Nz, 86
- Open, 346
- Pmt, 97
- PPmt, 98
- Rate, 98
- Replace, 103
- Right, 102
- Rnd, 95
- RTrim, 104
- Second, 93
- Space, 103
- Split, 103
- StrComp, 104
- Syd, 98
- Trim, 104
- Ucase, 102
- Year, 93
- математические, 94
- обратного вызова, 225
- преобразования типов, 80
- работы с датами, 88
- работы со строками, 100

Ц

Циклы, 119

- Do, 128
- For...Each, 151
- For...Next, 123

Э

Элемент управления, 63; 209

- группа переключателей, 236
- комбинированный список, 209
- простой список, 209
- свойство Tag, 239
- текстовое поле, 231

Научно-популярное издание

Майк Гандерлой, Сьюзан Сейлз Харкинз

Автоматизация Microsoft Access с помощью VBA

Литературный редактор	<i>П.Н. Мачуга</i>
Верстка	<i>О.В. Романенко</i>
Художественный редактор	<i>С.А. Чернокозинский</i>
Корректоры	<i>Л.В. Пустовойтова, В.В. Смоляр</i>

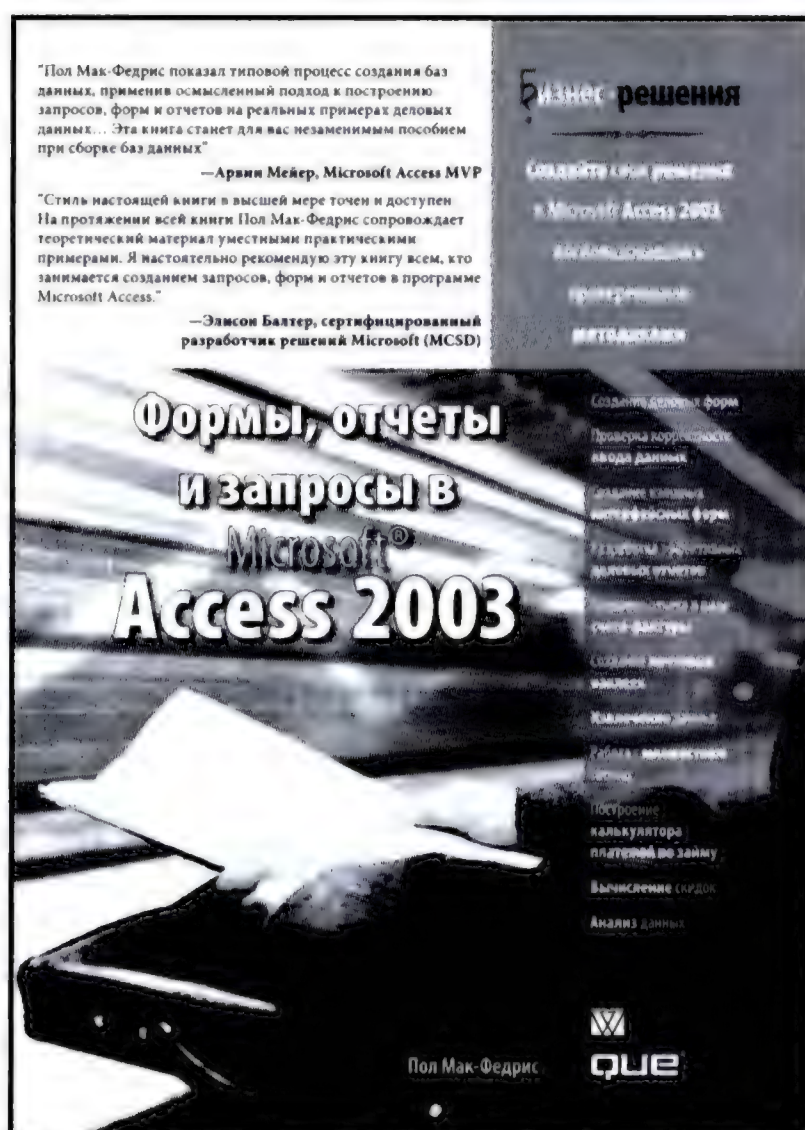
Издательский дом “Вильямс”.
101509, Москва, ул. Лесная, д. 43, стр. 1.

Подписано в печать 06.02.2006. Формат 70×100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 33,54. Уч.-изд. л. 19,2.
Тираж 3000 экз. Заказ № 523.

Отпечатано с диапозитивов в ФГУП «Печатный двор» им. А. М. Горького
Федерального агентства по печати и массовым коммуникациям.
197110, Санкт-Петербург, Чкаловский пр., 15.

ФОРМЫ, ОТЧЕТЫ И ЗАПРОСЫ В MICROSOFT ACCESS 2003

Пол Мак-Федрис



www.williamspublishing.com

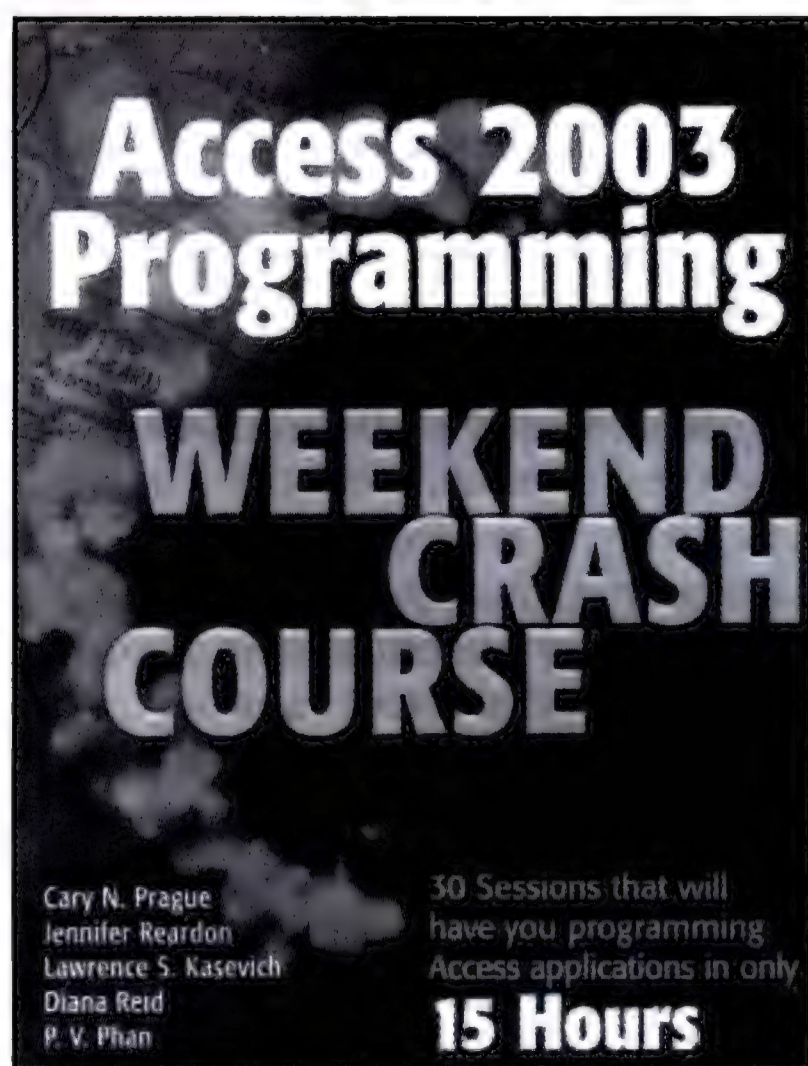
ISBN 5-8459-0814-0

Данная книга является особенной: в ней не делается попыток всеобъемлющего описания системы и изложения теоретических основ баз данных, для понимания чего требуется техническое образование. Вместо этого в книге приводится описание практических методик применения основных компонентов среды Access, воспользовавшись которыми можно создавать любые приложения для работы с базами данных. К таким компонентам относятся запросы, формы и отчеты. Первые главы каждой из трех частей книги дадут вам общее представление о рассматриваемых компонентах программы. В них вы узнаете о возможностях интерфейсов создания запросов, форм и отчетов. При этом будут рассмотрены практические примеры применения описанных методик. Последующие главы каждого из разделов расширят и углубят ваши знания в конкретных областях, что позволит создавать более сложные, а также специализированные запросы, формы и отчеты. Отдельная глава в каждом разделе посвящена особенностям работы с программой Access в реальных производственных условиях.

в продаже

ИНТЕНСИВНЫЙ КУРС ПРОГРАММИРОВАНИЯ В ACCESS 2003 ЗА ВЫХОДНЫЕ

*Керри Н. Праг,
Дженнифер Рирдон,
Лоренс Казевич,
Дайана Рид,
П. В. Фэн*



www.dialektika.com

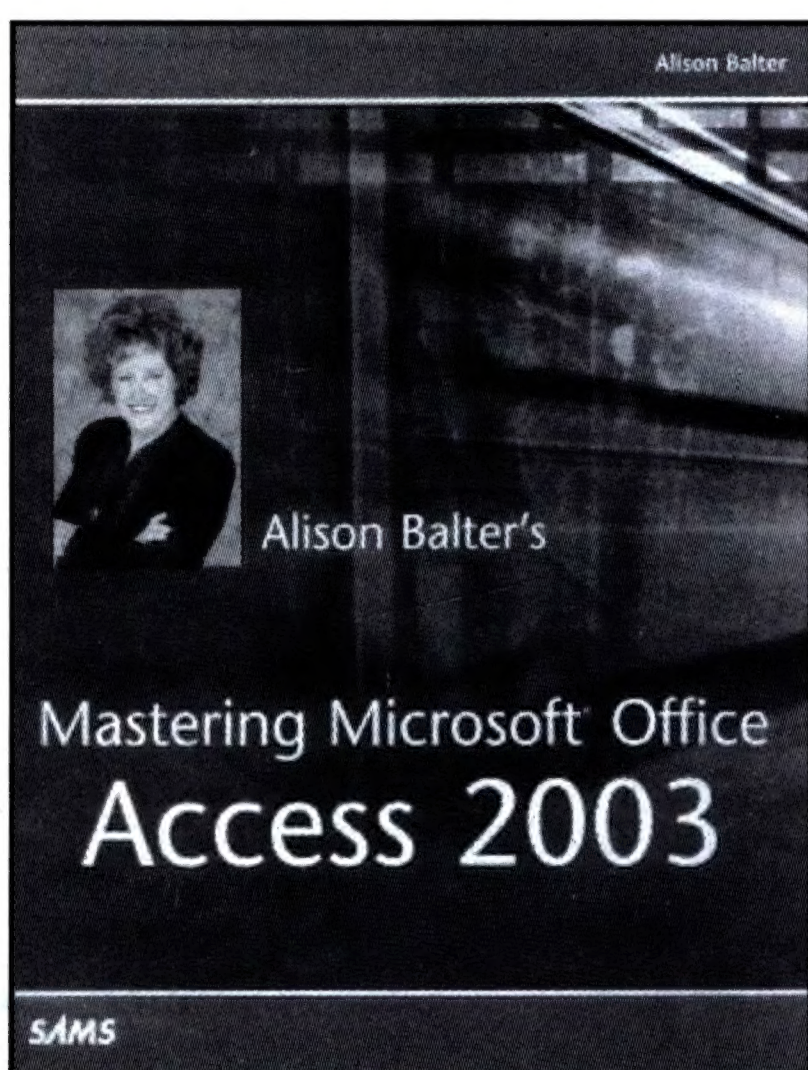
Книга посвящена вопросам программирования в среде Access 2003 — простой и мощной системы управления базами данных, входящей в состав новейшей версии общеизвестного комплекта программных продуктов офисного назначения Microsoft Office.

Авторы книги делятся своим многолетним практическим опытом щедро и непринужденно. Доступный стиль изложения, будет понятен даже новичкам, осваивающим передовые информационные технологии, а используемый пример приложения CheckWriter, который можно загрузить с сопровождающего Web-узла книги, добавляет наглядности.

4/2-26-4

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ В MICROSOFT OFFICE ACCESS 2003

Элисон Балтер



www.williamspublishing.com

Книга предназначена для пользователей, которым стал узок круг стандартных функций Access и макросов.

Вы научитесь создавать пользовательские программы для автоматизации базы данных, созданной с помощью Microsoft Office Access 2003, и узнаете, что многие рутинные функции по поддержанию базы данных могут выполняться автоматически, почти без вашего участия. В книге приведены примеры программных кодов, тщательно подобранные автором. Книга рассчитана главным образом на пользователей Access 2003 с различным уровнем подготовки, желающих расширить свой профессиональный кругозор и навыки программирования на VBA. Особенно полезной книга будет для разработчиков баз данных корпораций, доступ к которым необходимо обеспечить для группы пользователей локальной сети.



шения

Автоматизация Microsoft® Access с помощью VBA

Создайте свои решения в Microsoft
Access 2003, воспользовавшись
проверенными методиками

Книга *Автоматизация Microsoft Access с помощью VBA* поможет пользователю, работающему в Access, автоматизировать многие операции. Приложение Access включает в себя поддержку языка программирования VBA (Visual Basic for Applications); даже если до сих пор вы никогда не программировали, вы научитесь использовать VBA для увеличения продуктивности работы в Access.

Книга предназначена для профессионалов, которые используют Access в своей повседневной работе. Вы чрезмерно загружены работой, и у вас нет времени на изучение множества книг по компьютерной тематике? Эта книга поможет вам быстро приобрести основные навыки, необходимые для автоматизации баз данных.

Несмотря на то, что при написании книги использовалась программа Access 2003, изложенные в ней методики и концепции применимы практически ко всем версиям этого приложения.

В книге рассматриваются следующие вопросы:

- Основы работы в редакторе Visual Basic Editor
- Использование переменных, констант и типов данных
- Использование процедур
- Выбор встроенных функций VBA
- Работа с массивами
- Понятие объектов
- Работа с формами
- Анализ модели событий Access
- Работа с простыми и комбинированными списками
- Работа с отчетами
- Работа с коллекциями приложения
- Извлечение данных с помощью ADO
- Обработка данных с помощью ADO
- Расширенные операции с данными
- Работа с файлами данных
- Работа с файлами XML
- Обзор языка Jet SQL

Категория:	базы данных
Предмет рассмотрения:	Microsoft Access 2003
Уровень:	для пользователей средней и высокой квалификации



Посетите Издательский дом "Вильямс" в Internet по адресу:
<http://www.williamspublishing.com>
Посетите QUE в Internet по адресу: <http://www.quepublishing.com>



Майк Гандерлой — независимый программист и автор, проработавший в компьютерной индустрии более 25 лет. Он читает лекции на конференциях, составляет учебные курсы и пишет статьи по тематике, связанной с Access и другими базами данных, в том числе для журналов *Smart Access*, *Access-VB-SQL Advisor* и *Microsoft Office Solutions*. Майк Гандерлой тесно сотрудничает с командой разработчиков Access, он написал ряд программ для некоторых версий этого приложения. Из-под пера Майка вышло более двадцати книг, посвященных Access и другим вопросам работы с базами данных.

Сьюзан Сейлз Харкинз — независимый консультант по Access, автор многочисленных статей для технических журналов и других печатных и электронных изданий, в том числе *Element K Journals*, *PC Magazine*, *Access-VB-SQL Advisor*, *builder.com* и *devx.com*.

На Web-сайте:

Загрузите файлы баз данных, использованных в книге, по адресам
www.quepublishing.com
www.williamspublishing.com

ISBN 5-8459-0959-7



Бизнес-решения

Автоматизация Microsoft® Access
Помощью UVA

М. Гандерло
С. Харкинз



УДК